Secure Java Card for Federate Identity Management

Projet de diplôme 2008

David Olivier

Responsables internes : Philippe Joye, Rudolf Scheurer
Responsable externe : François Weissbaum
Expert : Pierre-Alain Mettraux
Etudiant : David Olivier

Lieu : Ecole d'ingénieurs et d'architectes de Fribourg, Suisse
Filière : Informatique

Table des matières

1 Introduction **4**

2 Cahier des charges **6**
2.1 Objectifs . . . 6
2.2 Tâches . . . 6
2.3 Milestones . . . 7
2.4 Planning . . . 7
2.5 Contraintes . . . 7

3 Analyse **8**
3.1 Description des technologies . . . 10
3.1.1 SSO . . . 10
3.1.2 OpenID . . . 10
3.1.3 SAML, Security Assertion Markup Language . . . 12
3.1.4 InfoCard . . . 15
3.2 Comparaison des technologies . . . 17
3.2.1 Architectures et utilisations différentes . . . 17
3.2.2 SAML vs OpenID . . . 17
3.3 Architectures . . . 25
3.3.1 "Strong Authentication", place de la Java Card . . . 25
3.3.2 LDAP vs Base de données . . . 30
3.3.3 Propositions d'architecture . . . 31

4 Spécification **37**
4.1 Applet JavaCard . . . 38
4.1.1 Fonctionnalités . . . 38
4.1.2 Structure . . . 39
4.2 Applet WEB Administration . . . 41
4.2.1 Fonctionnalités . . . 41
4.2.2 Communications (composants) . . . 41
4.2.3 Séquence de génération de carte . . . 42
4.2.4 Structure . . . 44
4.3 Applet WEB Authentification . . . 45
4.3.1 Fonctionnalités . . . 45
4.3.2 Communications (composants) . . . 46
4.3.3 Séquence d'authentification . . . 46
4.3.4 Structure . . . 47
4.3.5 Processus d'authentification . . . 48

4.4 Identity Provider . . . 50
4.4.1 Fonctionnalités . . . 50
4.4.2 Structure . . . 51
4.5 Service Provider . . . 57
4.6 Déploiement . . . 59
4.7 Base de données . . . 59

5 Implémentation 61
5.1 Applet JavaCard . . . 62
5.1.1 Contraintes . . . 62
5.1.2 Stockage des informations utilisateurs . . . 63
5.1.3 Génération de clé et traitement du challenge . . . 64
5.1.4 Sécurisation des cartes et de l'application (applets WEB) . . . 65
5.2 Applets Web Administration et Authentification . . . 67
5.2.1 Interface de l'applet WEB d'administration . . . 67
5.2.2 Interface de l'applet WEB d'authentifcation . . . 68
5.2.3 Contraintes . . . 69
5.2.4 Communication avec la carte . . . 70
5.2.5 Communication avec l'Identity Provider . . . 71
5.2.6 Transfert du pilote jct.dll et du fichier StyxWebSecurity.cap . . . 72
5.2.7 Installation de StyxWebSecurity.cap . . . 73
5.2.8 Traitement et processus . . . 74
5.3 Identity Provider . . . 75
5.3.1 L'interface . . . 75
5.3.2 Les servlets d'authentification et d'administration . . . 78
5.3.3 Logique métier . . . 81
5.3.4 Signature des applets WEB . . . 81
5.3.5 SAML . . . 82
5.4 Service Provider . . . 86
5.4.1 Interface . . . 86
5.4.2 Chargement des clés dans keytool pour SSL . . . 87
5.4.3 SAML . . . 88
5.4.4 Modification du toolkit SAML pour la signature des requêtes d'authentification . . . 89
5.5 Sécurisation des fournisseurs d'identité et de service . . . 92
5.5.1 Tomcat et SSL . . . 92
5.5.2 Politique de sécurité pour l'accès au pages . . . 92
5.5.3 Protection de l'accès au page de gestion . . . 93
5.6 Librairies . . . 95

6 Validation et tests de sécurité 96
6.1 Validation durant le développement . . . 97
6.2 Validation fonctionnelle . . . 102
6.2.1 Test des fonctions de l'Identity Provider . . . 102
6.2.2 Test des fonctions du Service Provider . . . 108
6.2.3 Test des fonctions de l'applet On-card et applet d'authentification ou d'administration . . . 110
6.2.4 Test du système dans sa globalité . . . 112
6.2.5 Résultats de la validation fonctionnelle . . . 114
6.3 Tests de sécurité du fournisseur d'identité avec la méthode OSSTMM . . . 115

6.3.1 Profil réseau . . . 115
6.3.2 StyxIdP, information sur le serveur . . . 115
6.3.3 Analyse de confiance . . . 119
6.3.4 Protection de la vie privée . . . 119
6.3.5 Résultats des tests . . . 120
6.4 Test d'accès à l'Identity Provider (faille dans le code) . . . 121
6.4.1 Principes . . . 121
6.4.2 Servlet d'administration . . . 121
6.4.3 Servlet d'authentificaton . . . 121
6.4.4 Servlet de téléchargement . . . 122
6.4.5 Pages de Management . . . 123
6.4.6 Pages accessibles . . . 123
6.4.7 Résultat des tests . . . 123
6.4.8 Corrections . . . 123
6.5 Visualisations des données échangées . . . 125
6.5.1 Authentification à l'Identity Provider . . . 125
6.5.2 Envoi d'une requête SAML depuis le SP à l'IdP . . . 126
6.5.3 Envoi d'une réponse avec authentification depuis l'IdP au SP . . . 127
6.5.4 Résultats . . . 127

7 Déploiement 128
7.1 Compilation et assemblage . . . 129
7.2 Installation et configuration . . . 131
7.2.1 Installation d'un fournisseur d'identité . . . 131
7.2.2 Installation d'un fournisseur de service . . . 131
7.2.3 Installation du client . . . 132
7.3 Mise en place des environnements de développement . . . 133
7.4 Les logiciels . . . 134
7.4.1 Eclipse et JCOP . . . 134
7.4.2 JavaScript Form Validation . . . 134
7.4.3 Jarsigner . . . 135
7.4.4 Keytool . . . 135

8 Conclusion 137

9 Remerciements 139

10 Figures 140

11 Bibliographie 143
11.1 Livres, revues, rapports . . . 143
11.2 Sites WEB . . . 143

12 Annexes 146
12.1 Plan du CD/DVD . . . 146

1 Introduction

Une des problématiques actuelles sur Internet est la prolifération des données d'authentification, car pour chaque service un utilisateur a un identifiant et un mot de passe différent. Cette surcharge de données oblige les utilisateurs à écrire ou stocker leurs données souvent de manière non sécurisée. Les systèmes de Federate Identity Management ont été inventés pour résoudre ce problème en permettant d'enregistrer les utilisateurs auprès d'un serveur et de lui déléguer l'authentification. Ainsi différents fournisseurs de service peuvent authentifier les utilisateurs en communicant avec leur serveur d'authentification respectif. Cet échange nécessite bien entendu une relation de confiance, ainsi qu'une authentification fiable sur le serveur d'identification.

Secure Java Card for Federate Identity Management est projet prospectif permettant d'établir le lien entre une technologie de sécurisation d'information (JavaCard) et une technologie d'infrastructure relative à l'authentification des utilisateurs (Federate Identity Management).

Il permet d'explorer les différents produits existants et technologies sur le marché afin d'établir la meilleure infrastructure et cohabitation possible entre la JavaCard et une technologie de Federate Identity Management. Cette étude est réalisée dans le chapitre d'analyse de ce rapport. L'aspect principal de cette analyse porte sur la découverte, l'évaluation et le choix d'une technologie de Federate Identity Management. Seul deux de ces technologies sont vraiment intéressantes, SAML [1] et OpenID [2].
La phase d'analyse consiste aussi à placer les JavaCard dans la technologie choisie pour augmenter sa sécurité, palier à ses faiblesses et faciliter son utilisation. La technologie JavaCard est déjà connu dans un précédent projet, "Java Card Security" qui consistait à explorer les

1. Security Assertion Markup Language http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
2. Identity System http://openid.net/

fonctionnalités utilisables lors du cryptage de fichier à l'aide d'un simulateur et non de carte physique.

La suite du projet est une spécification et implémentation d'un prototype aboutissant à l'illustration des concepts et contraintes exprimés durant la phase d'analyse. UML est le principal outil de spécification utilisé. En ce qui concerne l'implémentation des différents composants spécifiés, plusieurs outils et technologies sont nécessaires, car l'infrastructure du prototype est complexe et composée de différents éléments : serveur fournisseur d'identification, serveur fournisseur de service, applets web permettant de communiquer avec les cartes à puce et finalement application à l'intérieur de la carte à puce.

La validation du prototype est effectuée dans le chapitre qui lui est consacré grâce principalement à des tests fonctionnels. Elle comprend aussi des tests de sécurité selon la méthode OSSTMM [3] et des tests personnalisés selon l'applicatif.

Finalement le dernier chapitre aborde le déploiement : environnement de développement, compilation, création des packages, installation des serveurs, déploiement et installation du prototype. C'est une sorte de manuel d'installation qui permet de mettre en place facilement les différents éléments du système.

3. Open Source Security Testing Manual, manuel de test de sécurité open source, http://www.isecom.org/osstmm/

2 Cahier des charges

Le cahier des charges a pour but de définir les objectifs, les principales tâches et contraintes du projet. Le planning en fait aussi partie et définit le plan de travail, la durée de travail pour accomplir chaque tâche. Il permet d'estimer et de diriger le projet durant son déroulement.

2.1 Objectifs

Voici les objectifs du projet Secure Java Card for Federate Identity Management.

1. Concevoir et développer une application dont l'architecture profite de l'utilisation d'une JavaCard pour augmenter la sécurité d'une solution adéquate de Federate Identity Management.
2. Effectuer des tests de sécurité selon la méthode OSSTMM afin d'identifier les points faibles de l'application.

2.2 Tâches

- Créer une bibliographique
  - Se documenter sur FIM : SAML, OpenID, InfoCard et autres concurrents
  - Se documenter sur les possibilité entre JavaCard et Browser
- Évaluer des FIM
- Définir une architecture entre FIM et JavaCard (peut-être plusieurs possibilités)
- Modéliser une application "on et off card" se basant sur cette architecture peut inclure les tâches suivantes :
  - Chercher et sélectionner les fonctionnalités (UML use case)
  - Fournir une architecture (UML déploiement, composant)
  - Définir les divers éléments (UML classe)

  - Définir les divers états (UML états)
- Choisir et installer une plateforme de développement (JCOP [1] ou Cosmo [2])
- Développer l'application modélisée (peut-être en plusieurs versions)
- Valider l'application développée (test fonctionnel)
- Analyser les risques de sécurité avec la méthode OSSTMM
  - Définir des scopes
  - Analyser
- Fournir un rapport complet et une documentation sur l'application

2.3 Milestones

Voici les dates importantes relatives au projet :
- 15.09 Rendu du planning
- 24.09 Rendu du document d'analyse
- 24-30.09 Déterminer un rendez-vous pour le choix du FIM et de l'emplacement de la JavaCard
- 12.11 Rendu du rapport (18h)
- 13.11 Préparation poster pour 12h
- 14-15.11 Exposition
- 20-21.11 Défense

2.4 Planning

La planning est en annexe.

2.5 Contraintes

Les contraintes techniques sont définies dans les objectifs : Il faut absolument intégrer la technologie JavaCard, et il faut aussi utiliser une technologie de Federate Identity Management. Sinon aucune autre contrainte a été définie par les professeurs, responsable externe et expert. Le projet Secure Java Card For Federate Identity Management a clairement été définit comme prospectif et assez libre, ceci permettant des décisions durant son exécution comme le choix de la technologie de FIM ou le choix d'intégration de JavaCard.

1. Plateforme de développement de JavaCard fournie par IBM et reprise par NXP
2. Plateforme de développement de JavaCard fournie par Oberthur

3 Analyse

Introduction

La phase d'analyse consiste dans un premier temps à une recherche et une évaluation des systèmes de Federate Identity Management présents sur le marché. En second lieu, une proposition d'architecture permettant au moyen de la technologie JavaCard de sécuriser le système choisi précédemment.

Les systèmes de Federate Identity Management qui vont être présentés et comparés dans ce rapport sont : OpenID, SAML et InfoCard. InfoCard n'est pas un système en soi, mais un concept. Une InfoCard est une identité digitale. Plusieurs solutions emploient ce concept : Windows CardSpace [1] et Bendit [2]. OpenID est un standard qui a été inventé pour empêcher les spams sur les blog en identifiant les utilisateurs avant qu'ils puissent déposer leurs commentaires. SAML est un standard de communication ne servant pas qu'à la SSO [3] comme OpenID. Ces 3 "standards" sont supportés par des multinationales ce qui démontre leur importance sur le marché et qui assure leur pérennité.

La technologie JavaCard servira à sécuriser les faiblesses de la technologie choisie. Les JavaCard sont beaucoup plus que des simples SmartCard [4] qui stockent des informations. Elles permettent d'exécuter des programmes à l'intérieur d'une machine virtuelle qui leur est propre et dont elles garantissent la sécurité. Le système de "Strong authentication" est un concept exploré dans cette phase.

1. Méta système d'identité http://msdn.microsoft.com/fr-fr/netframework/aa663320.aspx
2. Communauté disposant de divers outils opensource pour la gestion d'identité http://www.bandit-project.org/
3. Abréviation de Single Sign-on expliqué plus en détails dans ce chapitre
4. Carte à puce au sens général, dans ce rapport carte mémoire sans microprocesseur

Elle se compose d'une description des différentes technologies, de leur comparaison, de l'apport de la technologie Java Card dans le projet et finalement de propositions d'architectures impliquant les JavaCard et les technologies de Federate Identity Management.

3.1 Description des technologies

Les technologies présentes dans ce chapitre sont principalement utilisées pour résoudre le problème du "single sign-on", SSO. Ce chapitre le décrit brièvement ainsi que les technologies précédemment citées.

3.1.1 SSO

Single Sign-On : Principe d'authentification permettant à un utilisateur d'utiliser un seul identifiant unique pour se connecter à plusieurs sites WEB différents (ou applications WEB). Les buts principaux sont la simplification de la gestion de mot de passe et de la gestion des données personnelles pour l'utilisateur.

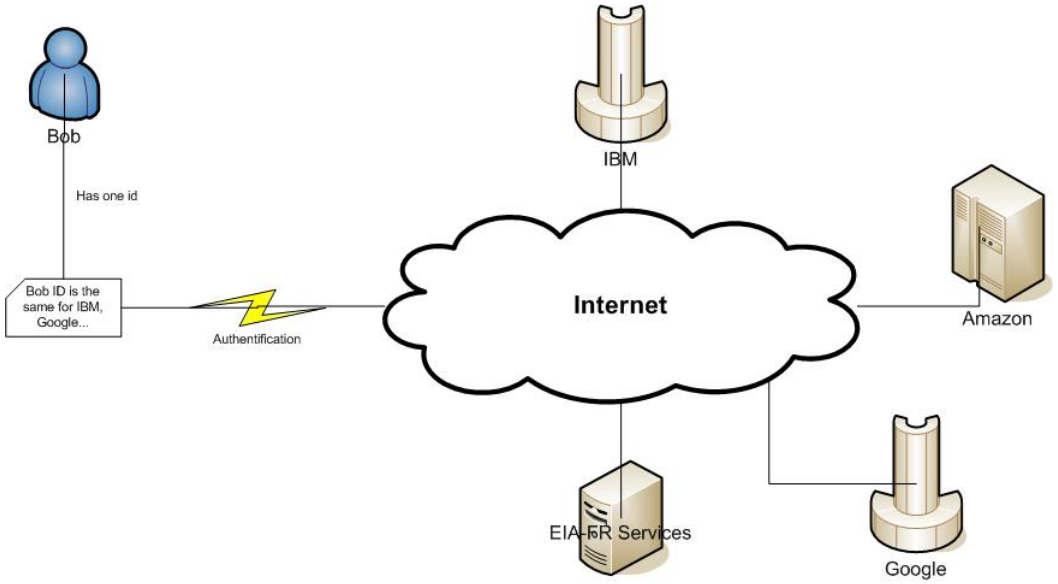

FIGURE 3.1 – SSO single sign-on

3.1.2 OpenID

Principe de fonctionnement

OpenID n'est pas seulement une syntaxe de message (comme SAML), mais s'est un protocole complet dont le but est le SSO. Dans ce sens il est beaucoup plus contraignant que SAML. Son avantage principal, c'est qu'il est immédiatement inter-opérable. Ce n'est pas le langage qui est compatible mais aussi la manière d'utiliser ce langage. N'importe quel Identity Provider est utilisable avec n'importe quel client (aucune relation de confiance ne doit préalablement être établie entre les Identity Provider et Service Provider). Voici le déroulement standard d'une authentification avec OpenID :

1. L'utilisateur envoi au "Relaying Party" le "User-Supplied Identifier" pour initialiser le processus de login. Cela se passe à travers un formulaire de login hébergé sur le RP (Relaying Party).

2. Relaying Party demande le document XRDS [5] avec le protocole Yadis [6] auprès de l'identifieur fourni par l'utilisateur à l'étape 1.
3. (Optionnel) Relaying Party crée et partage une clé avec l'Identity Provider en utilisant Diffie-Hellman Key Exchange [7]. Cela permet au Relaying Party de vérifier les signatures du OpenID Identity Provider. Cela enlève aussi la nécessité de requêtes pour vérifier la signature après chaque requête et réponse d'authentification.
4. Relaying Party redirige le navigateur WEB vers l'Identity Provider.
5. L'utilisateur est redirigé vers l'Identity Provider, s'identifie (la manière n'est pas définie dans OpenID), et complète la relation de confiance.
6. L'Identity Provider redirige l'utilisateur (avec une preuve cryptographique de son identification et les données personnelles qu'il a choisi de transmettre) vers la Relaying Party.
7. L'utilisateur est redirigé et est maintenant identifié à la Relaying Party, après vérification de sa part.

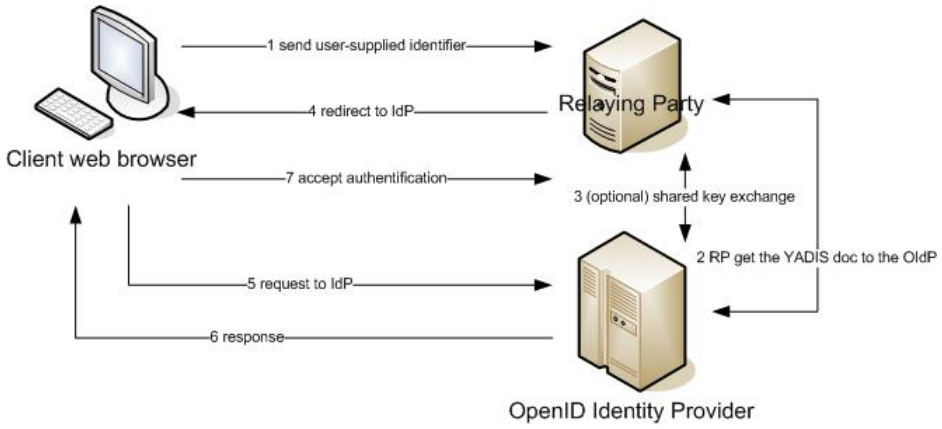

FIGURE 3.2 – Identification OpenID

Format des données Les messages de protocole sont du texte encodé en UTF-8 [8] convertis en bytes. La forme du texte est, par ligne, une paire de clé-valeur, séparée par une colonne. Les clés-valeurs sont signées pour s'assurer de leur authenticité.

Type de communication Il y a plusieurs types de communication qui sont développées dans les points suivants :

- Communication directe : initialisée par le RP vers OIIdP (OpenID Identity Provider).
  - Requête : Elles sont encodées dans le corps d'un POST.

5. eXtensible Resource Descriptor Sequence, document de méta données sur une ressource WEB
6. Service discovery system
7. Echange de clé sécurisé entre deux entités
8. Format de codage de caractère

  - Réponse : Elles sont encodées dans le corps d'une réponse HTML au format "text/plain" avec le code 200 en cas de succès et avec un code 400 en cas d'échec (mal formée ou arguments invalides).
- Communication indirecte : les messages qui sont passés par le navigateur du client et qui sont initialisés par le RP ou le OIIdP.
  - HTTP Redirect : utilisé pour passer à travers le navigateur du client.
  - HTML FORM Redirection : la redirection est en fait automatisée par un Java script.
  - Indirect Error response : ces messages sont utilisés en cas de requête mal formée ou contenant des valeurs incorrectes.

Détails Les détails relatifs aux types de message évoqués peuvent être trouvés facilement dans la spécification du protocole, "OpenID Authentication 2.0 [9]".

Utilisateurs

- IBM
- AOL
- BBC
- GOOGLE
- MICROSOFT
- MYSPACE
- ORANGE
- VERISIGN
- YAHOO

Librairies ou produits libres (java)

- OpenID4Java [10] : librairie permettant de créer un Identity Provider ainsi qu'un Service Provider.
- Joid [11] de Verisign : librairie permettant de créer un Identity Provider ainsi qu'un Service Provider.

3.1.3 SAML, Security Assertion Markup Language

Principe de fonctionnement

SAML est comme son nom l'indique un langage qui est utilisé pour des échanges d'informations relatives à la sécurité entre divers systèmes. Plusieurs profils d'utilisations sont possibles et décrits dans le langage. Les éléments importants de la spécification ainsi que le profil correspondant au SSO sont décrit en détails dans la suite de ce chapitre.

Identity Provider (IdP) Un IdP est responsable d'authentifier les utilisateurs. Normalement, il est présent dans chaque application et possède diverses fonctions selon son implication dans le logiciel. La plupart des systèmes complets maintiennent leurs propres bases de données avec leurs propres informations. SAML permet de partager ces informations entre diverses applications de manière sécurisée. Le confidentialité des informations (Privacy) peut-également être gérée. SAML utilise des assertions pour valider l'authenticité et l'accès aux ressources.

9. http://openid.net/specs/openid-authentication-2_0.html
10. http://code.google.com/p/openid4java/
11. http://code.google.com/p/joid/

Service Provider (SP) C'est une application qui propose des services ou des ressources à des utilisateurs. Le SP redirige les utilisateurs vers le IdP pour leur authentification et attend leur retour authentifié (à l'adresse qu'il a fourni au IdP). Une application peut avoir les 2 fonctions (SP et IdP).

Assertions Objets XML qui fournissent des informations par rapport à une entité. Plusieurs classes d'information existent :

- Déclarations d'authentification (Authentication statements) : Ils sont créés par le IdP qui authentifie l'utilisateur. Ils donnent des informations à propos de l'identification, du statut, de la validité et de la manière d'authentification.
- Déclarations d'attributs (Attribute statements) : Ils décrivent plusieurs caractéristiques à propos de l'utilisateur authentifié.

Protocols Les protocoles définissent les requêtes et les réponses entre les IdP et les SP. Plusieurs protocoles sont définis dans SAML :

- Authentication Request Protocol
- Single Logout Protocol
- Assertion Query and request Protocol
- Artifiact Resolution Protocol
- Name Identifier Mapping Protocol
- Name Identifier Management Protocol

Le protocole utilisé en premier pour le SSO est "Authentication Request Protocol". Il permet à un SP de faire une requête "authentication assertion" à propos d'un utilisateur particulier à un IdP.

Bindings Les "Bindings" décrivent comment les protocoles sont transportés entre les divers intervenants. Voici les "Bindings" que SAML définit :

- HTTP Redirect Binding : Spécifie comment transporter des messages de protocole avec des HTTP 302 redirect response.
- HTTP POST Binding : Spécifie comment transporter des messages encodés en BASE64 avec une HTML Form dans un HTTP POST.
- HTTP Artifact Binding : Les artifacts sont des identifiants. Le "Binding" correspond au HTTP Post et Redirect.
- SAML SOAP Binding : Spécifie comment transporter des messages de protocole avec avec SOAP 1.1 sur HTTP.
- Reverse SOAP (PAOS) Binding : Utilisé par les passerelles WAP pour qu'un client HTTP soit un SOAP responder.
- SAML URI Binding : Permet d'obtenir une SAML assertion en résolvant un URI.

Les "bindings" utilisés pour la SSO, dépendant du déploiement choisi, sont les 3 premiers : HTTP Redirect, HTTP Post et HTTP Artifact.

Profiles Un profil rassemble les assertions, protocoles et bindings pour effectuer un cas d'utilisation. Des contraintes ont été introduites dans les profils afin de promouvoir l'interopérabilité entre les différentes implémentations. Dans ce projet nous allons nous concentrer sur un des profils fourni par SAML : **Web SSO Profile** [12]. Les autres profils ne sont pas comparables avec OpenID. Ce profil spécifie comment utiliser un navigateur Internet pour faire du SSO en utilisant le "Authentication Request Protocol" et les "SAML Responses messages and Assertions". Voici une procédure d'utilisation du SSO avec HTTP Redirect :

12. http://www.oasis-open.org/committees/download.php/4647/sstc-saml-bindings-2.0-draft-02.pdf

1. L'utilisateur effectue un requête au SP.
2. Le SP construit un objet SAML de type "Authentication Request Protocol". Cette requête inclue la description de l'entité qui la créée, l'adresse où envoyer la réponse (URL) et une variable d'état du relai. Le XML est compressé en utilisant l'algorithme DEFLATE et encodé en utilisant BASE64. Il est ensuite placé dans le "HTTP Location header" comme un paramètre de requête.
3. L'utilisateur redirige cette requête au IdP. Lors de la réception l'IdP décode en BASE64 et décompresse le XML avec DEFLATE. Il se charge d'authentifier l'utilisateur et crée un objet "SAML Response" pour signaler que l'utilisateur est authentifié correctement. Bien entendu le XML suit la même procédure DEFLATE et BASE64.
4. L'IdP redirige le token au browser.
5. Le browser la redirige ensuite vers le SP.

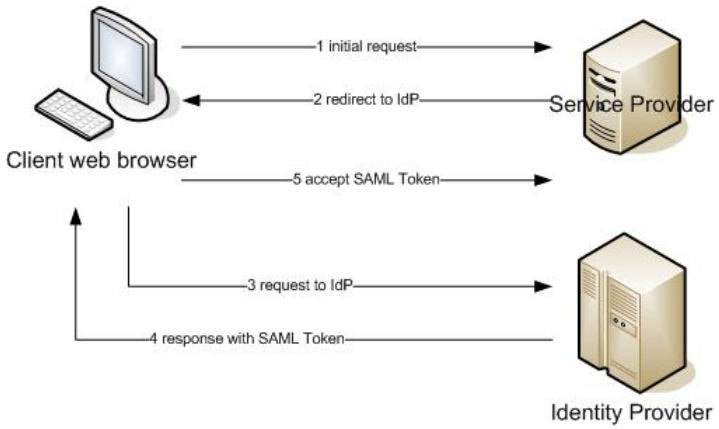

FIGURE 3.3 – Identification SAML

La procédure est la même avec HTTP POST sauf que la "SAMLRequest" est placée dans un champs caché d'un formulaire HTML. Il en va de même pour la réponse. Pour le "binding HTML Artifact", les requêtes et les réponses ne passent pas par le browser mais directement entre le SP et le IdP. La seule valeur qui passe par le browser est un artifact (petit identifiant). L'échange direct entre le SP et l'IdP est ensuite régi par le profil "Artifact Resolution Profile".

Détails Les détails relatifs aux "assertions", "protocols" et "bindings" peuvent être trouvés dans les documents relatifs présents dans la spécification de "SAML 2.0 [13]".

Utilisateurs

– Liberty Alliance [14] :

13. Sous références, http://en.wikipedia.org/wiki/SAML_2.0
14. http://www.projectliberty.org

- AOL
- Novell
- Sun
- Intel
- Oracle
- NTT
- franceTelecom
- ...

Librairies ou produits libres

- OpenSAML [15] : librairie bas-niveau permettant de gérer des objets SAML en Java.
- Clareity Security SSO [16] : framework basé sur OpenSAML pour créer des IdP et SP en Java.
- Shibboleth [17] : FIM complet basé sur OpenSAML en Java.
- Sun OpenSSO [18] : FIM complet utilisant SAML en Java.
- LASSO [19] : Librairie en C avec binding en Java permettant d'implémenter un Service Provider.
- Authentic [20] : Identity Provider en Python.
- SourceID SAML 1.1 Toolkit [21] : framework pour faire du SSO avec SAML en Java.

3.1.4 InfoCard

Définition du principe d'InfoCard

Digital identity représente une identité avec toutes ces informations. Il y en a 2 types, les gérées (qui sont obtenues par un Identity Provider SAML ou OpenID) et les autres (entrées à la main avec des informations personnelles).

Identity Selector Permet de regrouper les identités dans un gestionnaire qui autorise leur sélection lors d'une identification.

Windows CardSpace (seulement pour Windows)

Windows CardSpace est un programme qui stocke des "digital identities" et des références vers des identités gérées. Il permet aussi de sélectionner ces identités, c'est aussi un identity selector. Chaque identité est représentée par une "InformationCard". Il existe plusieurs librairies pour créer des Relaying Party acceptant des "InformationCard" pour Ruby, Java, C, CSHARP et PHP.

Principes :

1. Le site ou application web demande une identité.
2. L'application de choix d'identité apparaît.
3. L'utilisateur sélectionne l'identité qu'il veut.

15. http://www.opensaml.org/
16. http://code.crt.realtors.org/projects/websso
17. http://shibboleth.internet2.edu/
18. https://opensso.dev.java.net/
19. http://lasso.entrouvert.org/
20. http://authentic.labs.libre-entreprise.org/
21. http://www.sourceid.org/

4. Le logiciel CardSpace contacte l'Identity Provider correspondant pour obtenir un token signé qui prouve l'identité de l'utilisateur.
5. Communique avec le Relaying Party pour prouver l'identité de l'utilisateur.

Novell/IBM Bendit

Bendit est un projet qui contient plusieurs solutions open-source permettant de faciliter la gestion de l'identification :

- OTIS : Onramp to Identity Services permet d'accéder à des OTIS Identity Server.
- DigitalME : Identity selector pour Linux et MacOSX.
- Identity Provider : Identity provider permet de gérer les utilisateurs et leurs identifications.
- Higgins : framework permettant de mettre en place des identités digitales et des sélecteurs d'identités.
- OpenXDAS : est une implémentation de "OpenGroup's Distributed Auditing System".
- CASA : Composant permettant d'enregistrer les données confidentielles qui peuvent être utilisées pour le SSO par exemple.

Novell/IBM Higgins

Higgins est framework qui contient plusieurs composants permettant de faciliter le déploiement de solutions de gestion d'identités. Voici les solutions proposées par Higgins :

- Identity Selectors
  - GTK and Cocoa Selector, pour Firefox ou autre application sur Linux, FreeBSD and OSX.
  - RCP Selector, une application RCP Eclipse.
  - Firefox-Embedded Selector, pour Firefox sur Windows, Linux, and OSX.
- Identity Provider web services
  - STS IdP, WS-Trust Identity Provider (webapp and web service)
  - SAML2 IdP, SAML2 Identity Provider (webapp and web service)
- Relaying Party web site
  - Extensible Protocol RP Website 1.0, I-Card-enabled Relying Party site (webapp)
- Identity Attribute service
  - IdAS Solution, Identity Attribute Service uses Context Provider plugins to adapt existing data sources to the Context Data Model.

3.2 Comparaison des technologies

La comparaison des technologies sélectionnées n'est pas une tâche facile car elles diffèrent par leurs fonctions et leurs places au niveau des systèmes de Federate Identity Management. Dans le premier chapitre, OpenID, SAML et InfoCard vont d'abord être situées dans les systèmes de Federate Identity Management. Les chapitres suivants proposent une comparaison plus précise de SAML et d'OpenID sur ces divers points : perspectives pour les différents acteurs (utilisateurs, implémenteurs et déployeurs), confiance et confidentialité, sécurité et similitude des patterns de connexion.

3.2.1 Architectures et utilisations différentes

- SAML : Langage qui facilite la communication, profil SSO.
- OpenID : méthode complète (langage/protocole/technique).
- InfoCard : solution de stockage et de sélection d'identité.

Les 3 solutions se situent à des niveaux différents et sont difficilement comparables. SAML se trouve à un très bas-niveau (niveau langage des données échangées entre les différents intervenants). Il se peut donc que différents produits utilisant SAML ne soient pas directement compatibles même s'ils parlent le même langage car ils ne suivent pas forcément le même profil. OpenID quant à lui est une solution complète et concrète, contrairement à SAML, chaque Relaying Party peut communiquer sans adaptation avec un Identity Provider. L'interopérabilité est donc renforcer au détriment de la personnalisation fournis par SAML.

InfoCard ou plutôt les applications offrant cette technologie tel que Higgins ou Windows CardSpace se trouve à un autre niveau du côté du client. Cette technologie permet de stocker plusieurs identités digitales (qui peuvent être fournies par différents Identity Provider comme OpenID ou SAML mais aussi directement par l'utilisateur).

3.2.2 SAML vs OpenID

Perspectives au niveau de l'utilisateur

SAML est un framework abstrait et ne définit pas ce que l'utilisateur va voir ou faire contrairement à OpenID qui décrit les actions que l'utilisateur va faire. Il est possible d'utiliser SAML pour qu'il réagisse exactement comme OpenID.

Perspectives au niveau des implémenteurs

OpenID utilise des messages qui contiennent des données sous la forme de valeur-clé ce qui le rend plus simple que SAML qui utilise des fichiers XML pour communiquer les informations. Par contre OpenID est stricte dans sa syntaxe et son utilisation (comme le transfert des documents dans des requêtes HTML) tandis que SAML ne le précise pas. Les 2 standards sont implémentables avec beaucoup de langage et possède aussi beaucoup de librairies aidant à leur implémentation.

Perspectives au niveau du déployeur

OpenID est très simple a déployer et ne nécessite quasi aucune configuration. Il est supporté par beaucoup de script et est directement inter-opérable. Il suit le pattern, accepter tous les "incomers". Il est plus difficile de changer ce pattern pour faire évoluer un site avec des

sécurités supplémentaires, par exemple seul les membres de tel Identity Provider peuvent se connecter.

La grande différence avec SAML c'est qu'il est entièrement configurable par contre il est clair que l'inter-opérabilité en souffre car il faut définir des liens de confiance entre les différents Identity Provider et l'usage des données qu'ils contiennent. SAML est donc plus lourd à mettre en place mais beaucoup plus personnalisable.

Comparaison technique

Critère	**OpenID**	**SAML**
Logiciel ou librairie open source	disponible	disponible
Style de spécification	monolithique, seulement pour SSO	2 modules : assertions, requête-réponse. Ils sont utilisés dans des profils
Design	fixe, décentralisé et inter-opérable entre RP et IdP	complètement configurable, plusieurs profils autre que SSO
Structure des message	paire clé-valeur en "plain/text"	assertions et message de protocole en XML
Profitable	SSO concret	spécification abstraite qui peut être utilisé dans d'autres cas
Extensible	difficilement (seulement nouvelle clé-valeur)	facile grâce à XML
Confiance et sécurité	n'y apporte pas d'attention	regarder dans les profils d'utilisation
Lien au protocole	seulement HTTP GET et POST	aussi SOAP et autres protocole décrit dans les "bindings"
Identification	basé sur URL	dépend de l'implémentation

Comparaison au niveau de la confiance et de la confidentialité

Pour OpenID, le modèle de confiance par défaut est de faire confiance à tous le monde, contrairement à SAML qui spécifie que toutes les entités devraient faire les bonnes choses en respect avec la sécurité.

"OpenID Identifier" est un pointeur vers l'OpenID Identity Provider de l'utilisateur et nécessite que l'on fasse confiance à toutes les "Relaying Party" avec lesquelles il est utilisé. Selon certain spécialiste, c'est une invitation au phishing [22]. Du côté de SAML, si les mécanismes de découverte d'Identity Provider ne sont pas implémentés et que des relations de confiance sont créées entre les SP et IdP, il n'y a pas ces problèmes de phishing et de confiance dans les SP. SAML a donc un avantage certain dans ce domaine, même si l'inter-opérabilité en est fortement réduite.

22. technique utilisée par des fraudeurs pour obtenir des renseignements personnels dans le but de perpétrer une usurpation d'identité, http://fr.wikipedia.org/wiki/Phishing

Comparaison au niveau de la sécurité

Généralités OpenID prévoit des mécanismes de sécurité qui ne sont pas obligatoires et qui sont donc à la charge des implémenteurs et développeurs. Ils permettent de sécuriser et par conséquent de vérifier l'établissement des clés et la signature des messages. L'utilisation de SSL/TLS est aussi conseillée. SAML quant à lui, prévoit dans tous ses profils ainsi que ses bindings des mesures de sécurité. Tous les messages échangés sont aussi signés (xmldsig [23]). Plusieurs organismes ont évalué la sécurité des 2 standards, sur ce point SAML semble avoir une longueur d'avance, car la prise en compte de ses évaluations et leurs intégrations dans le standard semble être mieux organisées.

Phishing La constitution d'OpenID a un grand désavantage dans ce domaine car c'est l'utilisateur qui indique au moyen de son OpenID Identifier l'adresse du OpenID Identity Provider. L'utilisateur doit avoir confiance dans la Relaying Party à laquelle il soumet son identifiant car c'est elle qui peut l'induire en erreur en le redirigeant vers un faux Identity Provider. L'association entre le RP et IdP est basé sur Diffie-Hellman anonyme pour l'échange des clés, on ne peut donc pas vérifier réellement à qui on parle et il est possible qu'on subisse une tentative de phishing. Avec SAML il est conseillé d'employer la signature des messages basé sur les clés publiques, car cela permet de combattre les tentatives de phishing.

Signature des messages OpenID ne signe pas tous les messages et par conséquent ceux qui ne le sont pas peuvent être modifiés sans que l'on s'en rend compte. La seule défense d'OpenID fasse à ses attaques est d'utiliser TLS/SSL pour protéger les transmissions entre RP, IdP et navigateur de l'utilisateur. Mais cela ne protège pas entièrement la transmission, car même si les connexions sont sécurisées, elles traversent toujours le navigateur qui les décryptent et qui peut les modifier (browser cracking). SAML de son côté signe tout les messages par conséquent aucunes modifications de leurs contenus est possible, de plus TLS/SSL est aussi employé pour sécuriser les transmissions se qui rend SAML plus sécurisé qu'OpenID.

Les différents pattern d'authentification que OpenID et SAML peuvent réaliser

Il existe plusieurs patterns communs aux 2 standards que nous allons décrire dans les sous-chapitres suivants.

23. Spécification conjointe du W3C et de l'IETF utilisant XML pour contenir les signatures électroniques. On la désigne plus généralement sous le nom de XML Signature

SSO avec association
Avec OpenID

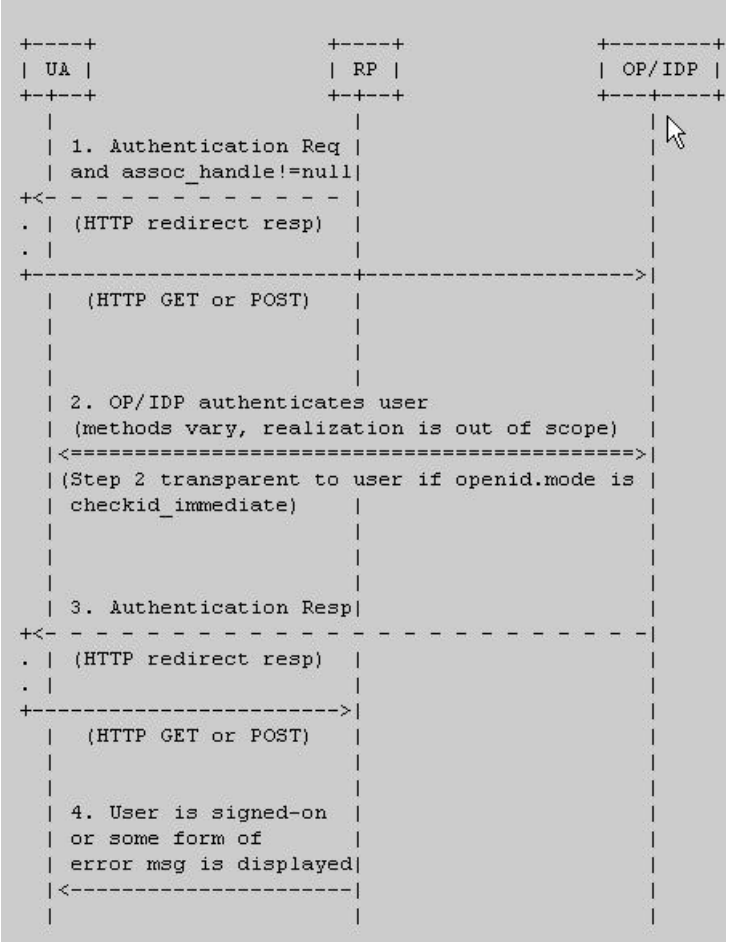

FIGURE 3.4 – OpenID Authentification with esthablished Association, J.Hodges www.identitymeme.org

Ce schéma se situe après la demande du client (navigateur) d'une ressource à la Relaying Party. La Relaying Party demande alors par l'intermédiaire du browser à l'Identity Provider l'identification de l'utilisateur. Ensuite l'Identity Provider se charge de l'authentification et renvoi la réponse à la Relaying Party. La phase de réponse à l'utilisateur par la ressource demandée est aussi omise sur ce schéma. UA signifie User Agent (navigateur WEB), RP signifie Relaying Party (fournisseur de service nécessitant une identification) et OP/IDP signifie OpenID Identity Provider (fournisseur d'identification).

Avec SAML

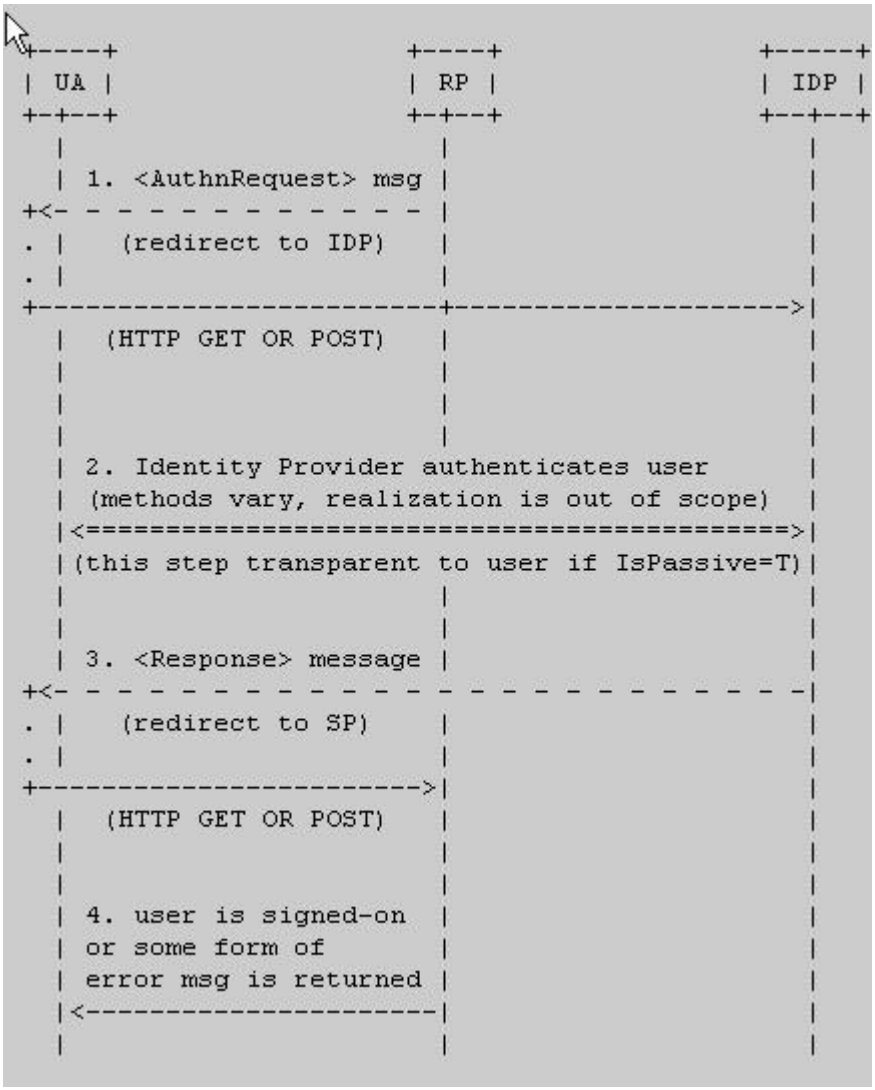

FIGURE 3.5 – SAML HTTP Redirect or POST based Web Browser SSO Profile, J.Hodges identitymeme.org

On remarque qu'avec SAML la procédure est exactement la même qu'avec la précédente qui utilisait OpenID. La phase de demande et de réponse sur une ressource du Service Provider est aussi omise. SP signifie Service Provider. On peut faire apparaître la similitude suivante : à Relaying Party correspond Service Provider.

SSO sans association établie
Avec OpenID

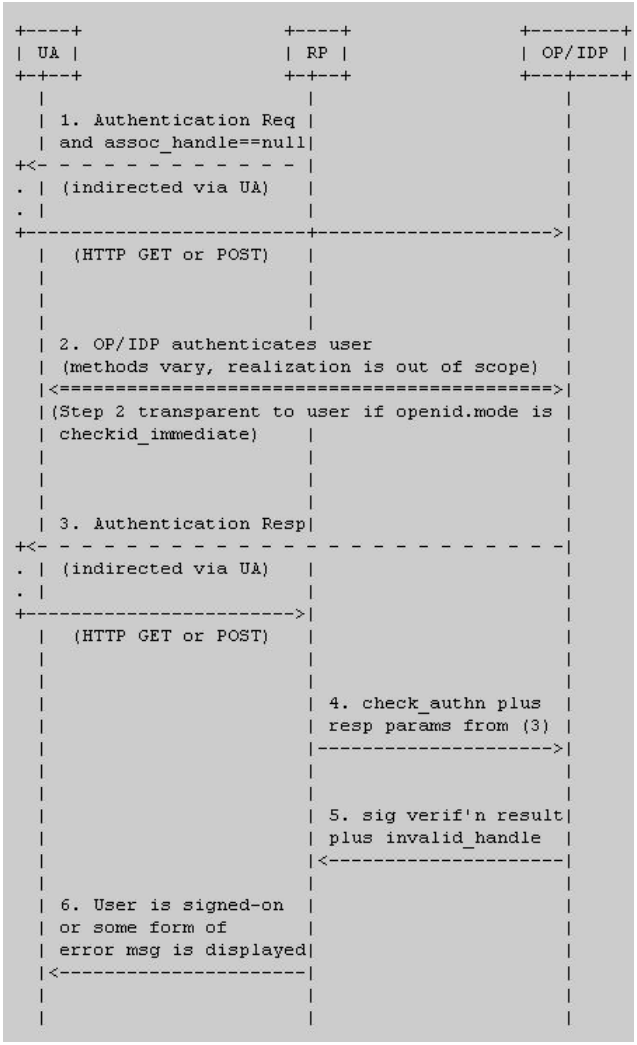

FIGURE 3.6 – OpenID Authentification without Established Association, J.Hodges identity-meme.org

S'il n'y a pas d'association préalable, on remarque que l'Identity Provider et la Relaying Party doivent communiquer pour que la RP s'assure que la réponse de l'IdP soit valide. Cette communication se produit après la réception de la réponse de l'Identity Provider par la RP. Lors de l'étape 4, la réponse 3 est entièrement renvoyée pour vérifier la signature du message.

Avec SAML

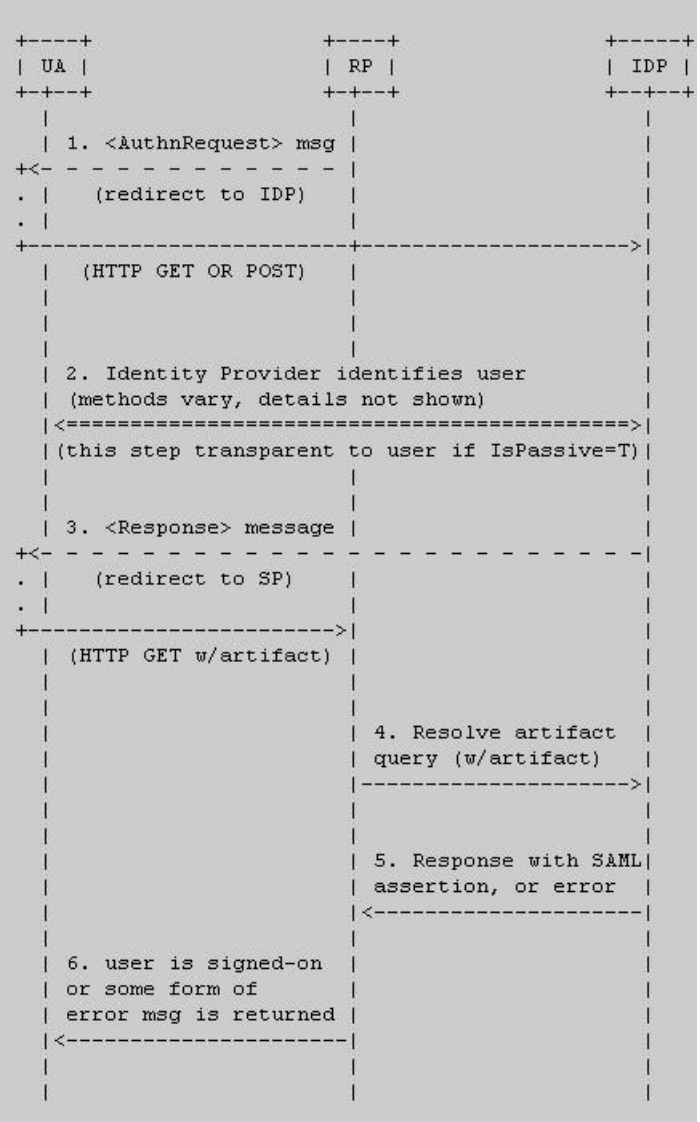

FIGURE 3.7 – SAML Web Browser SSO Flow with Artifact Binding used on Reply form IdP, J.Hodges identitymeme.org

On remarque que la procédure est de nouveau la même avec SAML qu'avec OpenID (expliqué précédemment). Dans ce cas SAML emploie "Artifact Binding" pour communique entre le SP et l'IdP. Cela consiste à dé référencer la "SAML Artifact" reçue au point 3 pour obtenir la "SAML Assertion" associée. Le RP vérifie ensuite les signatures de l'assertion.

Réponse non sollicitée (connexion direct au IdP puis redirection vers RP)
OpenID et SAML

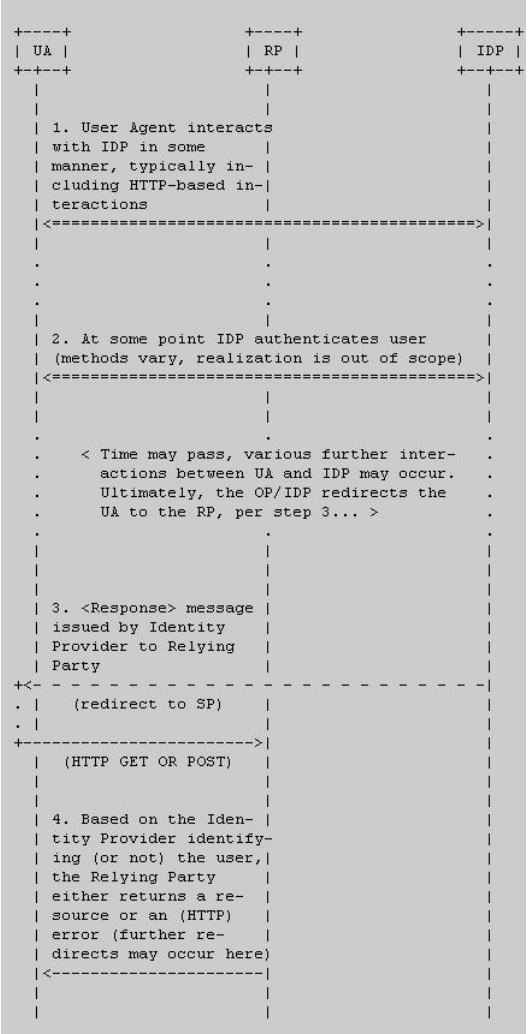

FIGURE 3.8 – SAML Web Browser SSO Profile "Unsolicited Response", J.Hodges identitymeme.org

Dans cette solution, l'utilisateur se connecte d'abord à l'IdP pour s'identifier et ce fait rediriger par la suite vers la Relaying Party. SAML supporte ce profil et le définit dans "OASIS SAML Profiles 2.0" mais OpenID est beaucoup moins éloquent à son sujet. Pour plus d'information à ce sujet : SAML and OpenID comparison de Hodges sur "identitymeme.org".

3.3 Architectures

3.3.1 "Strong Authentication", place de la Java Card

L'authentification forte[24] est une procédure d'identification qui requiert au moins deux facteurs d'identification parmi les suivants :

- Ce que l'utilisateur connaît : mot de passe, nip, ...
- Ce qu'il détient : carte, RFID, clé USB, soit une authentifieur (Token).
- Ce qu'il est : empreinte, yeux...
- Ce qu'il sait faire : signature manuscrite, ...
- Où il se trouve.

Plusieurs techniques peuvent se prêter à l'authentification forte :

- One Time Password (OTP)
- Certificat numérique (PKI)
- Biométrie
- Authentifieur ou Token
  - OTP : Secret partagé entre le Token et le serveur d'identification
  - PKI : La clé privé est sécurisée dans la carte

Une authentification forte en utilisant la JavaCard est possible car l'utilisateur détient physiquement une carte (qui contient par exemple une clé privée) et connaît aussi un PIN.

La JavaCard est une plateforme sûre qui permet d'effectuer des opérations sensibles dans un système embarqué évitant ainsi les problèmes de sécurité liés à la machine hôte (machine à laquelle la JavaCard est connectée). Le concept de "Strong Authentication" consiste à utiliser une paire de clé dont une est privée (celle-ci ne sort pas de la carte) et dont l'autre est publique (stockée dans une base de donnée ou un LDAP avec la correspondance à l'utilisateur). Le but de la JavaCard est donc de protéger cette clé et de l'utiliser pour démontrer que l'utilisateur est bien qu'il prétend être. Ci-dessous voici une schéma extrait du site de Sun qui démontre le principe de base de ce pattern.

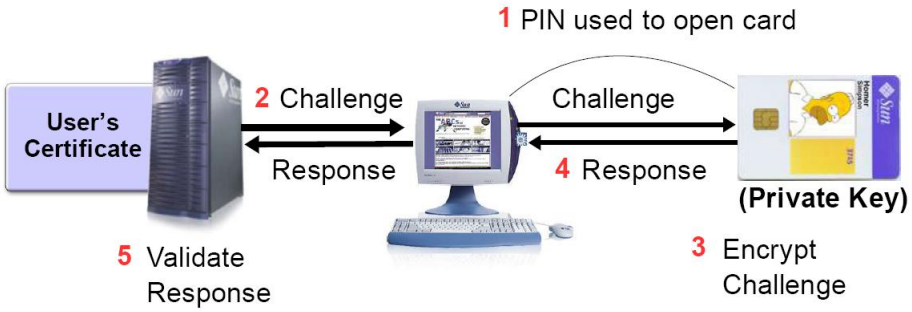

FIGURE 3.9 – Java Card Technology : Strong Authentication, Cryptographic Authentication with Java Card Technology Ellen Siegel and Matt Hill, JavaOne Sun Session TS-5251

24. Grandement inspiré de cette source http://fr.wikipedia.org/wiki/Authentification_forte

Les opérations nécessaires à la "Strong Authentification" sont les suivantes :

1. L'utilisateur a entré sa carte et compose le PIN.
2. L'Identity Provider crée un "challenge" (données aléatoires cryptées avec la clé publique du possesseur de la carte) et envoi du "challenge" vers la carte.
3. La carte décrypte le "challenge" avec sa clé privée et le crypte avec la clé publique de l'Identity Provider.
4. Le challenge de réponse est envoyé à l'Identity Provider.
5. L'Identity Provider, après décryptage, contrôle si le challenge correspond bien à celui qui l'a envoyé. Si oui, l'utilisateur est authentifié.

La double encryption évite que le challenge soit transmis en clair entre l'Identity Provider et la carte et renforce la sécurité. Si le challenge est transmis en clair, c'est le générateur de challenge qui peut être exposé car on sait que l'aléa n'est jamais vraiment aléatoire. Si l'on ne crypte pas le challenge envoyé à la carte et que la carte le crypte avec sa clé privé, on peut obtenir des couples "challenge-challengecrypté" qui sont très intéressant car ils peuvent être ré-employés si le générateur de challenge est pauvre.

Intégrer la technologie Java Card lors d'une identification à travers un navigateur Internet peut-être réaliser par plusieurs techniques :

- Applet WEB : Une application se charge dans le navigateur et accède au driver d'accès du lecteur de carte sur le PC hôte. Après quelques tests, il est possible de certifier qu'on est capable d'accéder au drivers du lecteur de carte et à la carte avec RMI depuis un applet web présent dans une archive jar signée.
- Plugin Mozilla : Une application contenu dans un jar dans un plugin Firefox permet d'accéder au driver du lecteur de carte. Cette méthode n'a pas été testé car elle limite l'utilisation de la solution à Firefox.
- Application autonome : Une application autonome se charge de l'authentification avec l'Identity Provider.

L'utilisation des systèmes de Federate Identity Management sert premièrement à simplifier la vie des utilisateurs. Il va de soi qu'il ne faut pas rajouter une couche de complexité par l'installation d'une partie cliente sur le poste qui doit s'identifier. Même si l'installation d'un plugin Firefox est simple, un applet est automatique et ne nécessite aucune connaissance de l'utilisateur. D'un point de vue technique l'applet et le plugin Firefox sont assez semblables même si le plugin introduit une couche tampon entre l'interface XUL avec du Java script et le jar du programme. Par contre l'application autonome n'utilise pas la même architecture et devrait soit être fortement liée au navigateur web, soit elle-même être un navigateur web. Cela rajoute une couche de complexité mais favorise aussi la sécurité de l'application car elle n'est pas proprement exécutée dans le browser (Chaîne de certification est gérée par le browser et peut être corrompu par exemple).

Il existe plusieurs méthodes pour communiquer entre un applet et le serveur qui l'a téléchargé :

- HTTP : à travers les pare-feu, lent, pas forcément Java sur le serveur, il faut former des requêtes, il faut former des réponses, seul l'applet forme des requêtes.
- Raw Socket Connection : filtré par les firewalls, rapide, pas forcément Java sur le serveur, communication bidirectionnelle, programme plus efficace du côté serveur, plus compliqué à coder.
- JDBC : pour communiquer avec un serveur de base de données

– RMI : permet l'invocation de méthodes sur un objet distant sur le serveur ou sur un objet distant sur l'applet. Peut passer à travers les firewalls à cause d'un transport sur le protocole http qui ralentit la connexion. Le serveur est obligé d'être écrit en Java.

L'utilisation de l'applet devient la solution la plus correcte par rapport au arguments énoncés précédemment. Cette utilisation sera développé à moins que des incompatibilités techniques surgissent et nous oblige à changer de voie.

JavaCard du côté de l'Identity Provider Il est aussi possible d'utiliser une JavaCard du côté de l'Identity Provider car il possède aussi une paire de clé qui est importante, et bien entendu la clé privé doit être protégée. Pour se faire on peut utiliser la carte du côté du serveur pour effectuer les opérations de cryptage des challenges envoyés aux cartes des utilisateurs. Par contre du côté du serveur, plusieurs requêtes peuvent arriver en même temps et dans ce cas une étude préalable de la charge maximale possible sur la carte et le nombre de requête maximale supportée (en même temps) devrait être effectuée. La carte ne permet pas d'accès concurrentiel et encore moins de processus interne, il faudrait donc stocker les demandes de challenge et exécuter leur cryptage une à une.

Code PIN sur JavaCard Il est possible de protéger un applet avec un code PIN représenté par l'objet OwnerPIN. Lors de l'entrée dans une méthode il faut tester si le code est valide pour la session par exemple :

```
// access authentication
if ( ! pin.isValidated() )
ISOException.throwIt(
     SW_PIN_VERIFICATION_REQUIRED);
```

Architecture avec gestion centrale Voici dans la figure suivante la place que les JavaCard peuvent prendre dans le projet. Chaque utilisateur en possède une qui contient sa propre clé privée qu'il ne partage jamais. Pour plus de sécurité, l'Identity Provider en a aussi une. Dans le cas de cette architecture la "strong authentication" est appliquée.

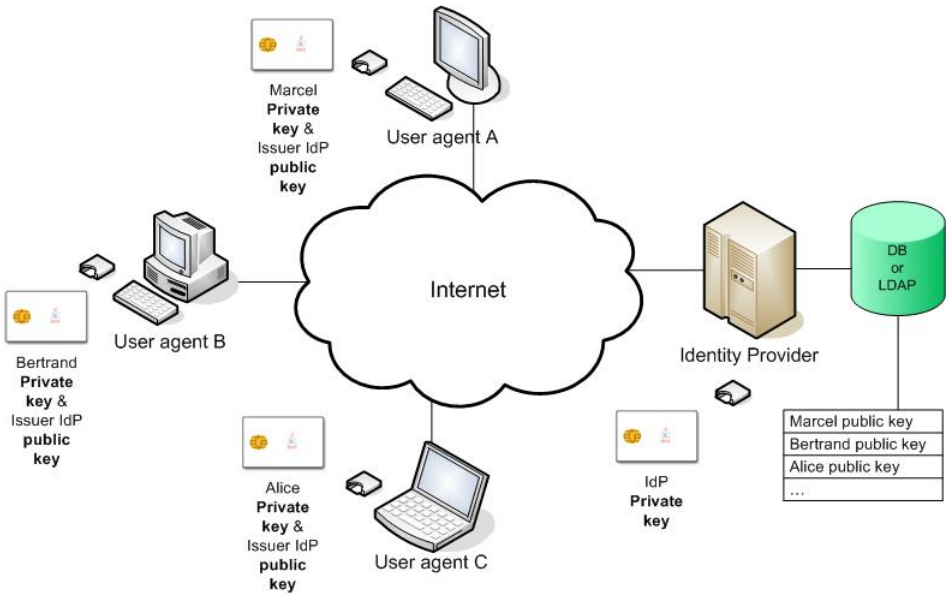

FIGURE 3.10 – Architecture avec gestion centrale

Architecture dé-centralisée La grande différence avec l'architecture précédente est qu'il n'y a pas de stockage des certificats par l'Identity Provider. Ils sont stockés sur la carte et signés par l'Identity Provider. Chaque carte possède donc sa paire de clé, dont la publique se trouve dans un certificat signé par l'Identity Provider, ce qui assure son identification.

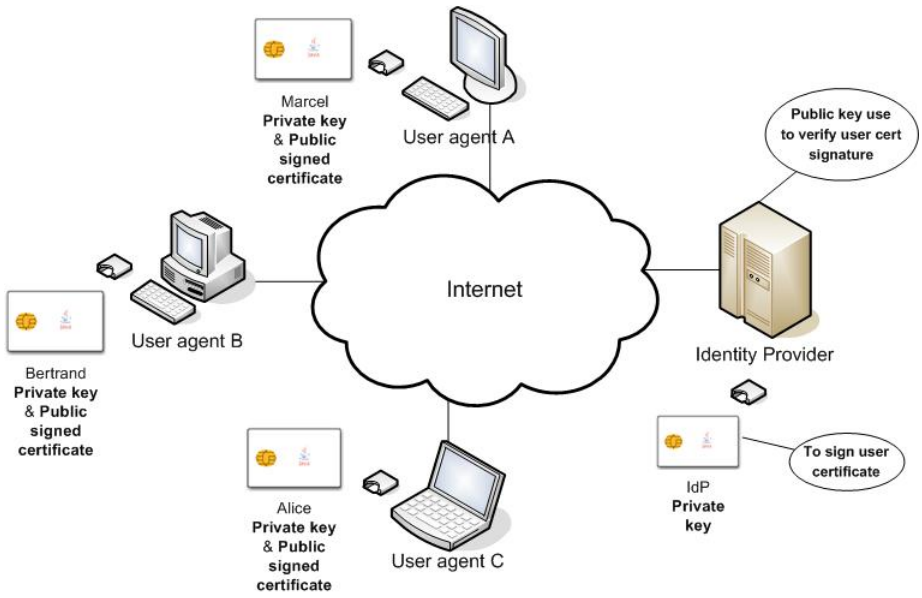

FIGURE 3.11 – Architecture dé-centralisée

Voici ci-dessous la procédure de création du certificat :

1. La carte envoi une requête de signature de certificat.
2. L'IdP signe le certificat avec sa clé privée.
3. L'IdP envoi le certificat signé à la carte.
4. La carte stocke le certificat.

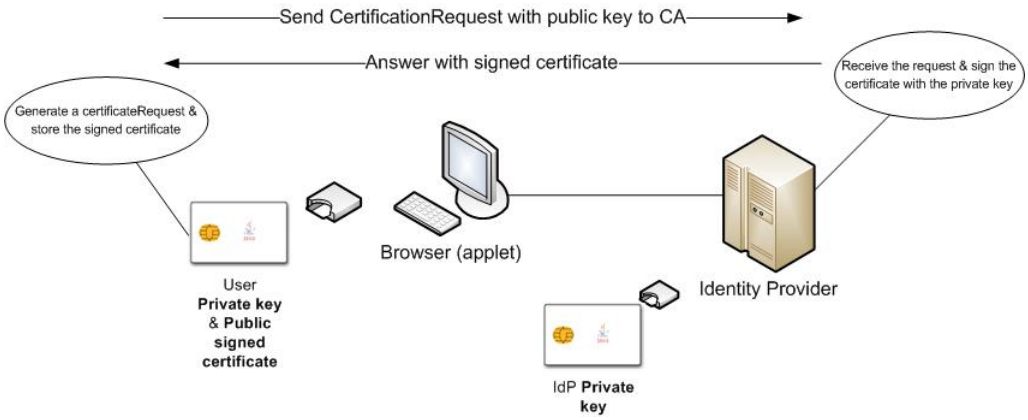

FIGURE 3.12 – Signature du certificat depuis la carte

Voici ci-dessous la procédure d'identification entre la carte et l'IdP :

1. L'IdP génère et un "challenge".

2. L'IdP envoi le "challenge" à la carte.
3. La carte récupère le "challenge" et le crypte avec sa clé privée.
4. La carte envoi le "challenge" crypté et son certificat publique signé par l'IdP (et sa clé privée)
5. L'IdP récupère le "challenge" et le décrypte avec la clé publique présente dans le certificat récupéré.
6. L'IdP contrôle la signature du certificat et valide ou non l'identification.

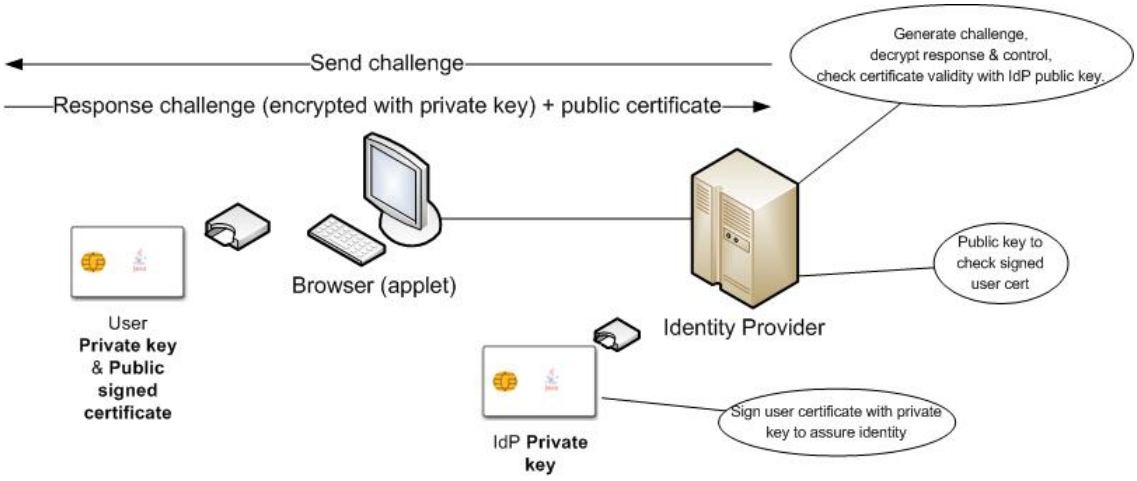

FIGURE 3.13 – Processus d'authentification avec le certificat signé sur la carte

3.3.2 LDAP vs Base de données

Le choix entre une solution basée sur la technologie LDAP [25] ou une basée sur une base de données au design propriétaire est influencé par les points suivants :

- La facilité de prise en main : Une base de donnée propriétaire est plus rapidement mise en place qu'un LDAP et ne nécessite pas de prendre en compte les différentes interfaces d'accès et d'utilisation d'un LDAP. De plus le design peut être facilement adapté par rapport au besoin de l'application.
- L'inter-opérabilité : Dans ce cas, la solution du LDAP est avantageuse car elle permet de s'intégrer dans une architecture existante sans trop de problème, mis à part la gestion des clés (privé/publique) et la création des certificats X.509 [26] qui est souvent propriétaire au serveur LDAP utilisé, et qui ne peut pas vraiment être fait sur la carte.
- La sécurité : Un produit en moins, équivaut aussi à ses failles de sécurité en moins, mais dans notre cas le LDAP est remplacé par une base de données, ce qui ne change rien. Par contre s'il on utilise une base de données, on peut se permettre de créer la paire de clés sur la carte et de générer nous même le certificat. Contrairement à celle-ci, LDAP offre directement la création de la paire de clé et ne permet souvent pas de rajouter ses propres certificats.

25. Lightweight Directory Access Protocol, protocole permettant l'interrogation et la modification des services d'annuaire, http://fr.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol
26. Certificat numérique pour l'échange de clé publique http://tools.ietf.org/html/rfc3280

La base de données est la meilleure solution si le temps est limité car la solution d'un LDAP comporte encore plusieurs risques :

1. Impossibilité d'importation de nos propres certificats publics ou impossibilité d'exportation de la clé publique présente dans un certificat dans le LDAP.
2. Impossibilité d'exportation de la clé privée, car la paire doit peut-être être générée dans le LDAP et non dans la carte. Cela remet en cause l'utilisation de la JavaCard.

Voici plusieurs serveurs LDAP (commercial ou non) :
- Apache Directory Server [27]
- Open Directory d'Apple [28]
- Critical Path Directory Server et Meta Directory Server [29]
- Fedora Directory Server [30]
- Red Hat Directory Server [31]
- OpenLDAP [32]
- Novell eDirectory [33]
- Sun Directory Server Enterprise Edition [34]
- OpenDS a Sun Open Source Directory Server [35]
- IBM SecureWay Directory [36]
- IBM Tivoli Directory Server (formerly IBM Directory Server) [37]
- IBM Lotus Domino [38]
- Active Directory de Microsoft [39]

3.3.3 Propositions d'architecture

Plusieurs architectures se sont profilées en utilisant les technologies étudiées dans les chapitres précédents, elles sont décrites dans les sous-sections suivantes.

OpenID Identity Provider (and Service Provider)

Cette architecture consiste à la création d'un Identity Server permettant d'authentifier un utilisateur à l'aide de la Java Card et de la "Strong authentication", expliquée au chapitre précédent. Vue qu'OpenID est un standard il n'est pas nécessaire de créer le système d'authentification du côté du Service Provider car plusieurs parties clientes sont déjà disponibles et permettrons d'utiliser notre Identity Provider. Pour les tests il sera néanmoins nécessaire d'avoir une à deux parties clientes afin de valider notre OpenID Identity Provider.

D'un point de vue technique, il est possible d'utiliser des librairies et des frameworks open source permettant de faciliter le développement d'un Identity Provider ou d'un Service Provider. OpenID4Java et JOID sont 2 frameworks qui peuvent le faciliter.

27. http://directory.apache.org/
28. http://www.apple.com/fr/server/macosx/technology/opendirectory.html
29. http://www.criticalpath.net/en/43/news/?news=186850
30. http://directory.fedoraproject.org/
31. http://www.redhat.com/directory_server/
32. http://www.openldap.org/
33. http://www.novell.com/products/edirectory/
34. http://www.sun.com/software/products/directory_srvr_ee/dir_srvr/index.xml
35. http://www.opends.org/
36. http://www.ibm.com/servers/eserver/iseries/ldap/schema/
37. http://www.ibm.com/software/tivoli/products/directory-server
38. http://www.ibm.com/software/fr/lotus/
39. http://www.microsoft.com/france/technet/produits/win2003/AD_ADAM.mspx

Voici une architecture possible en créant un OpenID Identity Provider et un OpenID Service Provider avec une des 2 librairies à disposition. L'Identity Provider peut utiliser comme source de données pour les utilisateurs soit une base de données privée qu'il faudra modéliser, soit un serveur LDAP capable de stocker et délivrer des certificats publiques (PKI). Cette architecture comprend aussi la mise en place du concept de "strong authentication" entre la Java Card et l'Identity Provider.

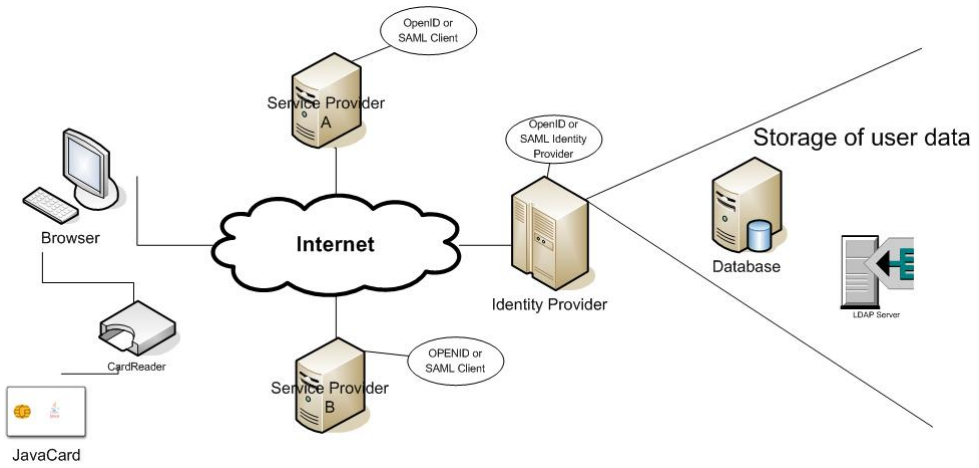

FIGURE 3.14 – Architecture globale pour OpenID ou SAML

SAML Identity Provider (and Service Provider)

Cette architecture consiste à la création d'un Identity Server permettant d'authentifier un utilisateur à l'aide de la JavaCard et de la "Strong authentication", expliquée au chapitre précédent. SAML permet de simplifier la gestion de l'authentification et peut-être employé par l'Identity Provider et le Service Provider. Vue que SAML est un standard il n'est pas nécessaire de créer le système d'authentification du côté du Service Provider car plusieurs parties clientes sont déjà disponibles et permettrons d'utiliser notre Identity Provider.

D'un point de vue technique, il existe plusieurs librairies, frameworks ou toolkits permettant d'implémenter plus facilement un Identity Provider ainsi qu'un Service Provider : OpenSAML, Clareity Security SSO, LASSO, SourceID SAML 1.1 Toolkit.

Voici une architecture possible en créant un SAML Identity Provider et un SAML Service Provider avec une des librairies à disposition. L'identity Provider peut utiliser comme source de données pour les utilisateurs soit une base de données privée qu'il faudra modéliser, soit un serveur LDAP capable de stocker et fournir des certificats publiques (PKI). Cette architecture comprend aussi la mise en place du concept de "strong authentication" entre la Java Card et l'Identity Provider.

Java Card Strong Authentication dans un produit libre

Plusieurs produits libres existent sur le marché. Certain implémente les 2 standards de SSO : SAML et OpenID. Le but de cette architecture serait de modifier l'authentification

d'un de ses produits pour qu'il utilise la "Strong Authentication" que des tokens Java Card peuvent offrir. Voici une liste de solutions open source qui semblent être les plus influentes sur le marché :

- Sun OpenSSO
- Shibboleth
- Alfa and Ariss OpenASelect

Du point de vue technique, cela dépend bien entendu du choix de la solution, car elles n'offrent pas toutes les même fonctionnalités et API de développement.

OpenSSO est une solution open source fournie par Sun sur la base d'un produit commercial : Sun Java System Access Manager. Elle est composée d'un API client pour les Service Provider (.NET, Java,...) et d'une partie serveur pour l'Identity Provider. Il est possible de créer des modules d'authentification supplémentaires. La documentation est abondante et possède même des exemples. Des API sont à disposition pour utiliser LDAP et les certificats.

Shibboleth est utilisé par les universités de beaucoup de pays et par SWITCH en Suisse. On peut développer des extensions pour ce logiciel qui permettent de modifier la façon dont un utilisateur peut s'identifier auprès de l'Identity Provider. Le code source et la documentation nécessaire sont disponibles sur le site officiel du projet.

OpenASelect[40] est une solution open source de Federate Identity Management qui remplace A-Select, une solution précédente qui est utilisée par le gouvernement Danois. Il est possible d'obtenir le code source et de modifier l'identification des utilisateurs. Néanmoins la documentation est moins abondante que pour Shibboleth et OpenSSO, de plus le projet ne paraît pas vraiment mature (manque d'informations disponibles). Ce produit n'est donc pas retenu pour une implémentation.

Java Card Safe for Digital Identity

Une Java Card est un espace sécurisé où les identités digitales pourraient être stockées et contrôlées (seulement les identités qui doivent se connecter à un Identity Server SAML ou OpenID). Plusieurs projets et applications existent sur le marché et stockent les identités digitales sur le disque de la machine hôte. Hors nous savons que la machine hôte n'est pas forcément une plateforme sûre, c'est pour cela qu'on pourrait augmenter la sécurité d'une de ces solutions en utilisant une Java Card pour stocker et utiliser les identités. Les 2 solutions existantes sont :

- Windows CardSpace
- Eclipse Higgins

Du point de vue technique, cela dépend bien entendu du choix de la solution, car elles n'offrent pas toutes les même fonctionnalités et API de développement, bien entendu Higgins est une solution open source contrairement à Windows CardSpace.

Concernant l'architecture, la carte sert de stockage de donnée pour l'application de gestion d'identité. Il n'y a qu'une machine qui comprend l'application (Higgins ou CardSpace), et un middleware qu'il faut réaliser pour stocker les identités digitales dans la carte. On peut aussi

40. http://www.openaselect.org/trac/openaselect/

imaginer porter à l'intérieur de la carte la fonctionnalité permettant de vérifier les identités digitales qui sont reliées à un Identity Provider.

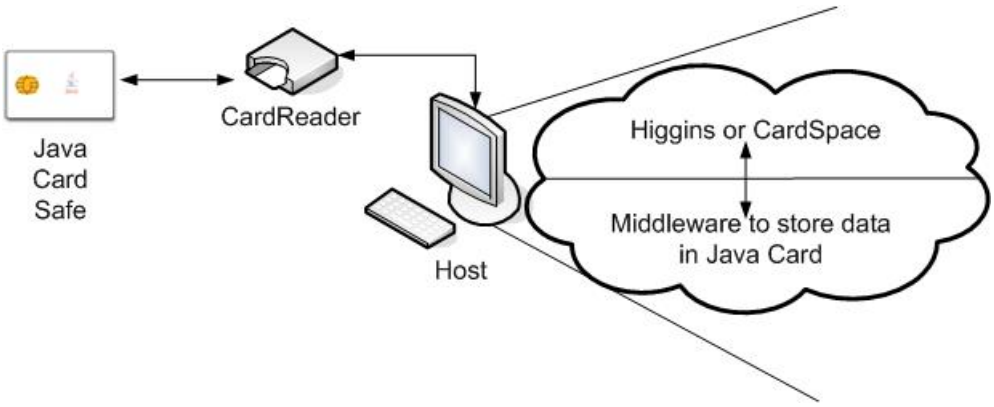

FIGURE 3.15 – Architecture pour "Java Card Safe"

Conclusion

Le choix de la technologie dépend fortement des attentes liées au projet. Les 2 technologies principales, OpenID et SAML, ont chacune leur force et leur faiblesse. Elles ont aussi leur propre contexte d'utilisation. OpenID sert à identifier, mais ne sert pas à interdire (toute personne qui s'authentifie à accès au service). Cela peut bien entendu être modifié mais plus difficilement que dans SAML, qui quant à lui, est beaucoup plus facilement personnalisable grâce à une spécification plus ouverte.
Les 2 systèmes se situent aussi à des niveaux différents. OpenID est directement visible (login avec URL), SAML quant à lui, laisse beaucoup plus de libertés aux développeurs. Il va de paire que si SAML est beaucoup plus libre, il est aussi plus dur à implémenter qu'OpenID, même si au niveau de l'utilisateur, la ressemblance est assez proche. Malgré cela, l'implémentation est grandement facilitée par la présence de framework libres pour les 2 technologies.
En lisant plusieurs avis d'utilisateurs et de déployeurs, ainsi quand analysant ces technologies, on arrive à la conclusion qu'OpenID est moins sécurisé et moins paramétrable que SAML. C'est aussi prouvé par des tentative réussi de phishing sur le système OpenID. Pour plus de flexibilité et de sécurité, il est recommandé d'utiliser SAML. Pour plus de simplicité et moins flexibilité et de sécurité, il est recommandé d'utiliser OpenID. La technologie InfoCard a aussi été analysé, mais est très difficilement comparable, car elle se trouve à une autre niveau et est plutôt complémentaire. Elle permet d'employer OpenID et SAML dans la gestion d'identités personnelles qu'elle propose. Il est aussi dommage d'utiliser une JavaCard comme un simple espace de stockage d'identités digitales.

Le choix de l'architecture dépend aussi fortement des attentes liées au projet. Néanmoins plusieurs contraintes sont nécessaires. Le déploiement d'une infrastructure complète comprenant au minimum un Service Provider (celui qui demande l'identification), un Identity Provider (celui qui authentifie l'utilisateur), une Java Card (qui permet l'identification sûr auprès de l'Identity Provider) et un fournisseur de données sur les utilisateurs (LDAP ou DB propriétaire). Un des choix qui doit être fait concerne soit l'implémentation d'un Identity Provider en utilisant un framework avec l'utilisation du concept de "Strong Authentication" sur une carte Java, soit l'intégration de ce concept dans une solution open source existante. Néanmoins, une solution construite depuis un librairie est plus adaptable qu'une solution existante même s'il est open source. De plus l'utilisation du protocole n'est pas automatique en utilisant une librairie et permet d'en avoir une meilleure connaissance. Un second choix concerne le stockage de la clé publique - elle-même stockée dans un certificat signé par l'Identity Provider. Il y a 2 possibilités soit dans la base de données ou LDAP du Identity Provider, soit sur la carte de l'utilisateur. Ce choix dépend d'où le client préfère stocker le certificat publique. Dans un cas ou l'autre, le stockage du certificat ne remet pas en cause la pertinence de l'identification.

La place de la Java Card dans le projet est une contrainte de taille car elle doit apporter un plus à une solution de Federate Identity Management. J'ai donc pensé intégrer un concept connu, "Strong authentication", à travers un navigateur WEB, pour renforcer la phase d'identification de l'utilisateur avec l'Identity Provider. A la fin de la phase d'analyse j'ai découvert un papier "Pluggin a Scalable Authentication Framework into Shibboleth" qui conforte mon analyse et les possibilités de la réaliser. Cet article parle d'un prototype pour l'authentification au moyen d'un Java Card qui a été conçu par l'université de Manchester en 2005. Dès lors les technologies WEB et Java Card ont évoluées et permettent certaines améliorations comme l'utilisation de RMI entre l'applet WEB et l'applet on-card à la place

de commandes APDU. La possibilité d'utiliser une Java Card du côté de l'Identity Provider servant aux tâches de cryptage n'est pas exclue mais n'est pas considéré comme primordiale. L'utilisation de cette même carte pour générer un challenge (données aléatoires) n'est pas raisonnable quand on connaît son manque puissance en la matière.

La solution retenue, après discussion avec les différents intervenants, est une architecture utilisant SAML avec une identification forte basée sur la technologie JavaCard.

Pour conclure, cette phase d'analyse a permis de découvrir et de connaître, à un niveau assez avancé, les "protocoles" OpenID et SAML pour continuer la spécification d'une solution les utilisant. Les solutions basées sur InfoCard quand-elles ne requiert pas l'utilisation d'une JavaCard, mais plutôt d'une simple Smartcard. C'est pour cela qu'elles ont naturellement été mises de côté, après leurs analyses.

4 Spécification

Introduction

La spécification s'est déroulée en même temps que l'implémentation à cause des inconnus liés aux technologies qui pouvaient intervenir dans l'architecture du prototype. Tous les différents composants sont traités dans ce chapitre : applet JavaCard, Applet WEB d'administration, Applet WEB d'authentification, fournisseur d'identité, fournisseur de service. L'infrastructure (déploiement) ainsi que la base de données sont aussi traités dans ce chapitre.

4.1 Applet JavaCard

Cette application est présente sur la carte, ses fonctionnalités sont très limitées et permettent de sécuriser les informations de son authentification auprès d'un fournisseur d'identité. L'authentification est aussi sécurisée à l'intérieur de la carte.

4.1.1 Fonctionnalités

Les fonctionnalités présentent dans le diagramme suivant ont été implémentées. Certaines fonctionnalités précédentes n'ont pas pu être implémentées faute de temps : création d'une CSR [1] sur la carte, stockage des certificats de l'utilisateur et de l'Identity Provider sur la carte (impossible de récupérer les clés publiques de ses certificats sur la carte dans les délais impartis).

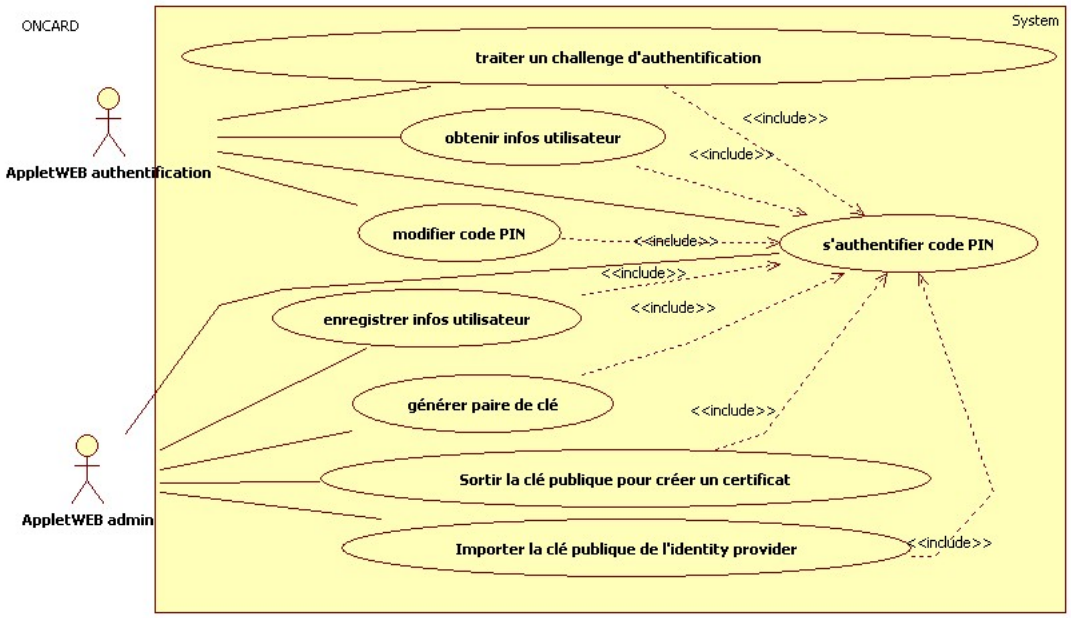

FIGURE 4.1 – Cas d'utilisations pour l'application on-card

Les fonctionnalités peuvent être regroupées en 2 groupes, celles qui servent à l'administration du système et celles qui servent à l'authentification de l'utilisateur. Les premières sont utilisées par l'applet d'administration et les secondes par l'applet d'authentification. Voici une description de chacune d'elle :

- Traiter un challenge d'authentification : lors de la procédure d'authentification expliquée dans le rapport d'analyse "Strong Authentication", on décrit l'utilisation d'un challenge qui est crypté (avec la clé publique de l'utilisateur) et envoyé à la carte. C'est à cet instant précis que le cas d'utilisation est actif car il décrypte avec sa clé privée

1. Certificate Signing Request, requête de signature de certificat, http://en.wikipedia.org/wiki/Certificate_signing_request

le challenge et le crypte ensuite avec la clé publique du fournisseur pour le renvoyer à l'expéditeur.
- Obtenir les informations de l'utilisateur : Dans la carte il ne suffit pas d'avoir une clé privée (celle de l'utilisateur) et un clé publique (celle du fournisseur d'identité). Mais il faut aussi une référence à l'utilisateur pour retrouver sont certificat sur le serveur d'identification ou ses informations personnelles. C'est dans ce but que certaines données sont stockées sur la carte. Un identifiant unique ainsi que le nom et le prénom correspondant à l'utilisateur sont donc stockés dans la carte.
- Modifier le code PIN : s'il on désire protéger l'accès à la carte par un code PIN, il est inutile de ne pas offrir la possibilité à l'utilisateur de le modifier.
- S'authentifier : l'utilisateur de la carte doit aussi pouvoir déverrouiller les fonctionnalités de la carte qu'il souhaite utiliser pour s'authentifier auprès du fournisseur d'identité. Pour simplifier, on peut dire que l'utilisateur à 2 niveaux d'authentification : au niveau de la carte avec son code PIN, et au niveau du fournisseur d'identité avec sa paire de clés (privée, publique).
- Enregistrer les informations de l'utilisateur : les informations de l'utilisateur (identifiant, nom, prénom) sont insérées à l'initialisation de la carte.
- Générer la paire de clés : les clés sont générées sur la carte. La clé privée ne sort jamais de la carte et n'est accessible que pour traiter un challenge (c'est à dire, le décrypter).
- Sortir la clé publique pour créer un certificat : ce cas d'utilisation est assez explicite, ce certificat sera ensuite stocké et utilisé par le fournisseur d'identité.
- Importer la clé publique du fournisseur d'identité : cette clé publique sera ensuite utilisée pour renvoyer le challenge de manière sécurisée au fournisseur d'identité.

4.1.2 Structure

Le diagramme de classe suivant indique la structure de l'application on-card qui est composée d'un applet qui se conforme à la StyxJCInterface. Celle-ci permet ensuite d'utiliser RMI [2] pour appeler les méthodes présentes dans la carte.

2. Java Remote Method Invocation, http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

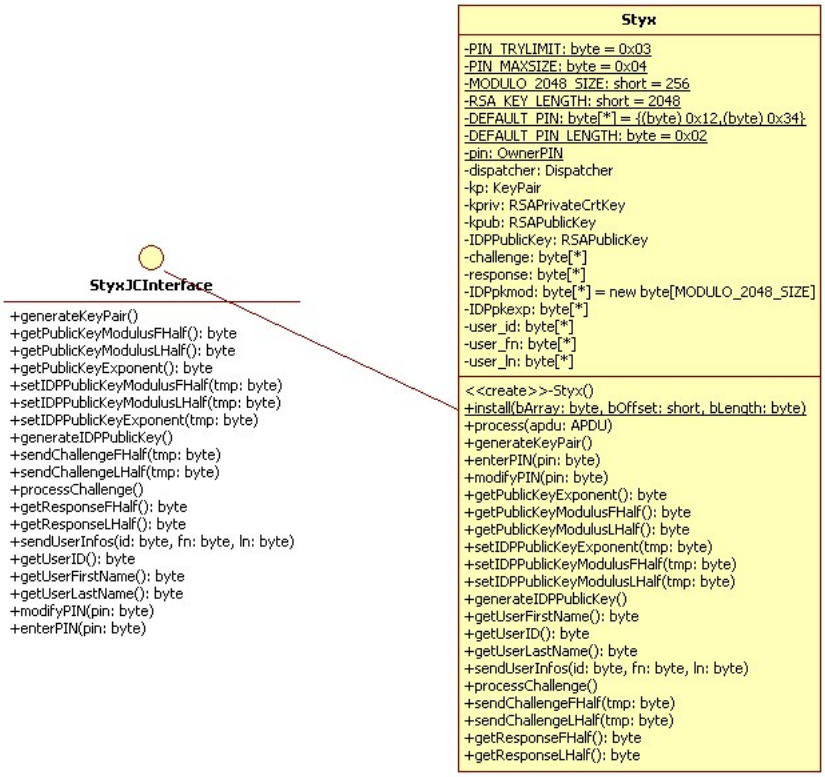

FIGURE 4.2 – Classes pour l'application on-card

Cette application a un structure relativement simple du fait du peu de fonctionnalités disponibles dans la plateforme de développement "NXP JCOP41 JavaCard 2.2.1".

4.2 Applet WEB Administration

4.2.1 Fonctionnalités

Cet applet WEB sert essentiellement de passerelle entre le fournisseur d'identité et la carte pour son initialisation. Les fonctions principales sont exposées dans le diagramme de cas d'utilisation suivant.

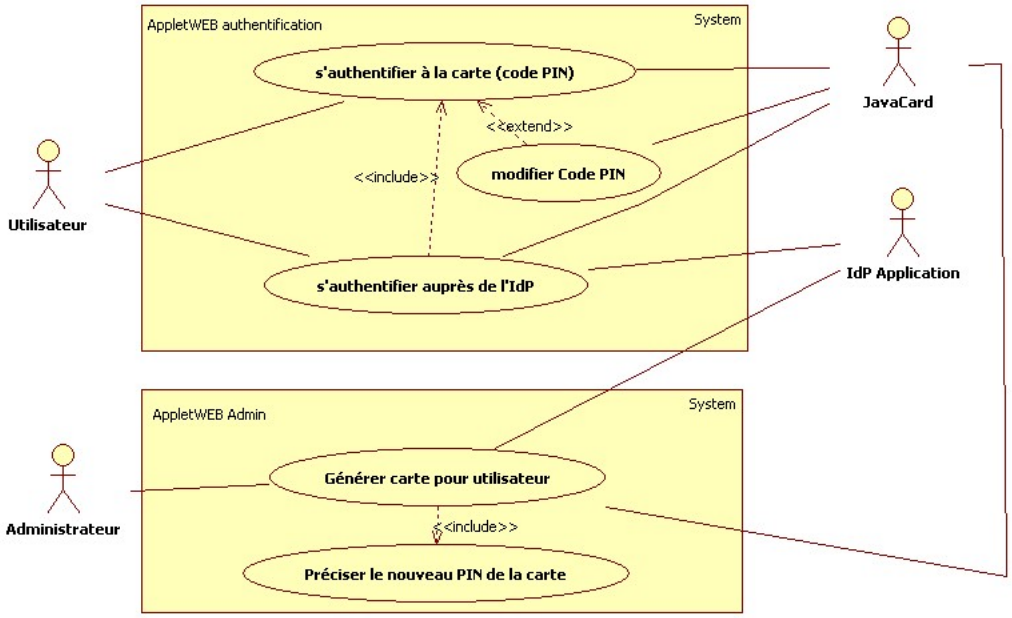

FIGURE 4.3 – Cas d'utilisations pour les applets

Voici les divers cas d'utilisations pour l'applet d'administration dans le détail :

- Génération de la carte pour l'utilisateur : ce cas d'utilisation est une procédure qui dépend de plusieurs fonctionnalités appartenant soit à la carte, soit au fournisseur d'identité.
  - Génération d'une paire de clé (dans la carte)
  - Obtention de la clé publique de la carte et création d'un certificat correspondant à son utilisateur sur le fournisseur d'identité.
  - Stockage des informations de l'utilisateur et de la clé publique du fournisseur d'identité dans la carte
- Préciser le nouveau code PIN de la carte : permet de personnaliser le code PIN de la carte à son initialisation pour plus de sécurité

4.2.2 Communications (composants)

Les communications entre l'applet WEB et la carte et entre l'applet WEB et le fournisseur d'identité ont été abstraites dans des classes gérant elles-même la connexion et l'interfaçage avec leur entité respective. Cela permet de cacher la complexité des différents transferts

de paramètres et de valeurs de retour ainsi que des différentes actions demandées pour un processus (transfert de clé, demande de challenge, etc). Voici ci-dessous les composants de la communication :

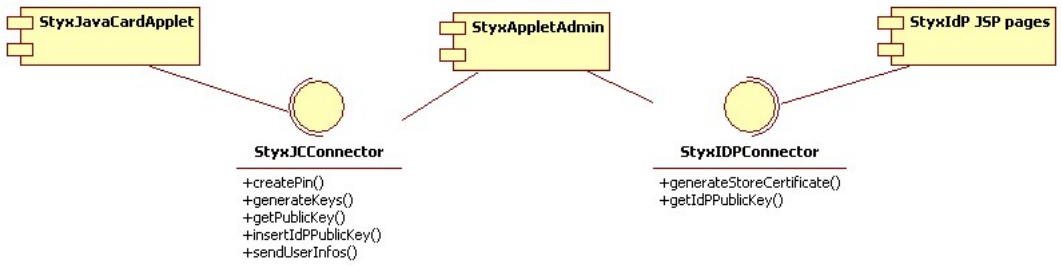

FIGURE 4.4 – Composants de communication, applet d'administration

4.2.3 Séquence de génération de carte

Deux séquences distinctes sont nécessaires à la génération des cartes. Tout d'abord il faut s'assurer d'installer l'applet dans la carte et de la rendre sélectionnable pour l'utiliser depuis l'extérieur. Cela se fait par des commandes APDU [3] et le téléchargement du fichier CAP [4], fichier contenant l'application à installer sur la carte. Dans un second temps il faut configurer la carte pour un utilisateur et la personnaliser, c'est ce que le diagramme suivant démontre.

3. Application Protocol Data Unit,http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4_annex-a.aspx

4. format de fichier contenant une application JavaCard

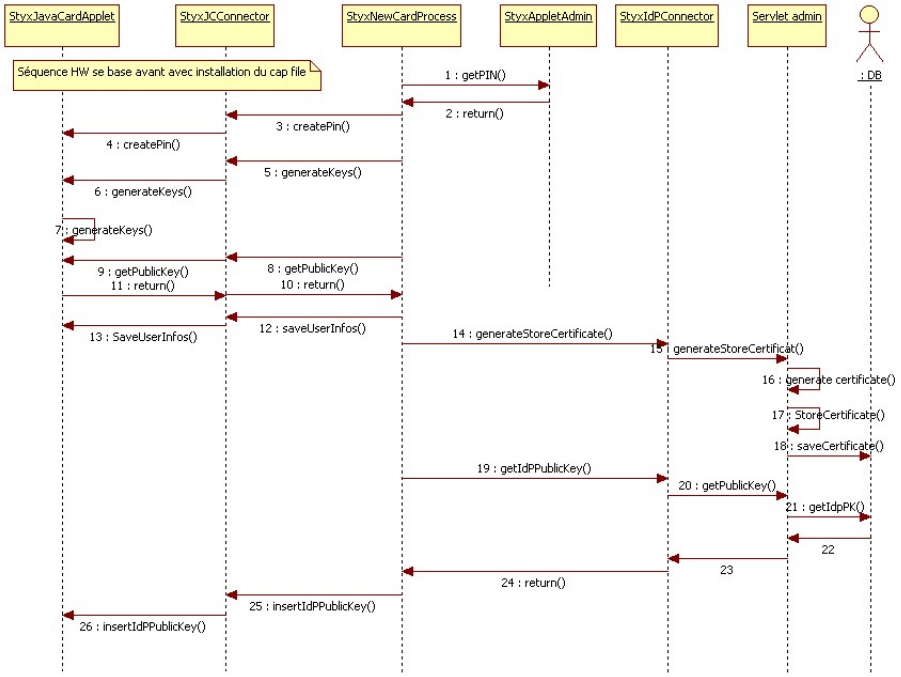

FIGURE 4.5 – Séquence génération de carte, applet d'authentification

Voici la procédure d'initialisation d'une carte :
- Il faut obtenir le PIN que l'administrateur veut donner à la carte.
- Il faut changer le PIN de l'applet "on-card" par celui fourni précédemment.
- Générer la paire de clé sur la carte.
- Télécharger la clé publique depuis la carte.
- Transférer les informations de l'utilisateur sur la carte.
- Générer un certificat (avec la clé publique précédente) sur le fournisseur d'identité.
- Stocker le certificat.
- Obtenir la clé publique du fournisseur d'identité.
- Insérer cette clé dans la carte.

Voici la séquence d'installation matériel d'une carte :
- Connexion à la carte.
- Connexion au CardManager [5].
- Authentification au CardManager (soit avec les clés JCOP, soit avec les clés STYX [6], et soit avec les clés fournies par l'utilisateur)
- Sécurisation du CardManager avec les clés STYX.
- Suppression des anciens packages et applet STYX.
- Téléchargement du CAP de l'application "on-card".
- Installation du package et de l'applet depuis le fichier CAP sur la carte.

5. Application présente dans toutes les JavaCards qui permet de gérer les applications (installation, suppression, ...) et la sécurité
6. Dénomination du prototype StyxWebSecurity

4.2.4 Structure

La structure de l'applet se compose des classes suivantes :
- StyxAppletAdmin : GUI de l'applet.
- StyxIDPConnector : Interface de connexion au fournisseur d'identité.
- StyxJCConnector : Interface de connexion à la carte à puce.
- StyxNewCardProcess : Processus se chargeant de l'initialisation d'une nouvelle carte.
- StyxStartActionListener : Gestionnaire d'événement permettant de lancer le processus StyxNewCardProcess.

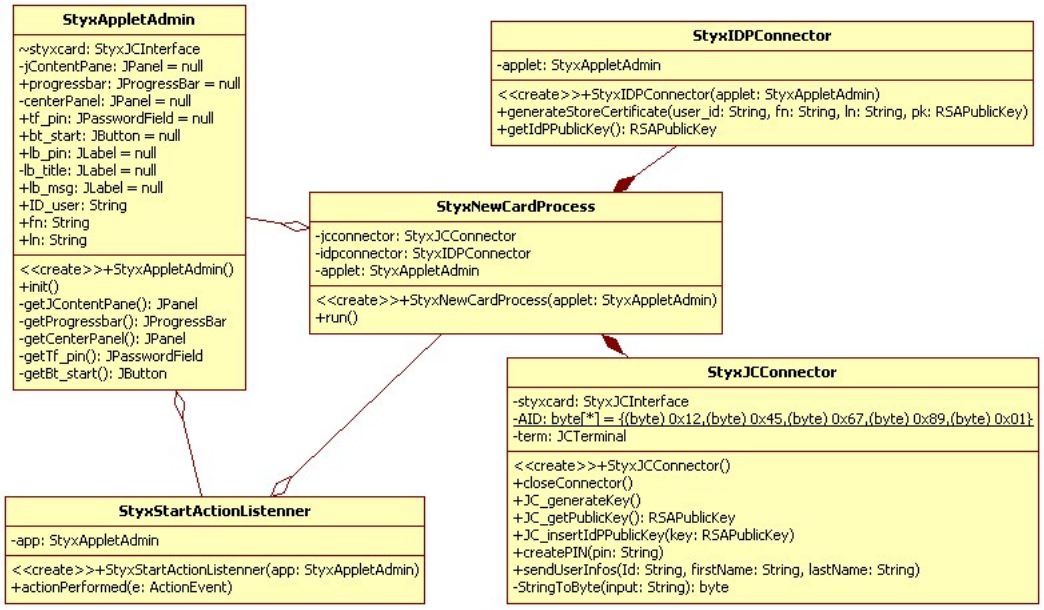

FIGURE 4.6 – Classes pour applet d'administration

4.3 Applet WEB Authentification

4.3.1 Fonctionnalités

Les fonctionnalités de l'applet authentification sont principalement utiles à l'authentification de l'utilisateur à la carte (Code PIN), et ensuite au fournisseur d'identité (Strong Authentication, challenge/response, 2 paires de clés). Une autre fonctionnalité qui est aussi nécessaire pour le confort des utilisateurs est la possibilité de modifier le code PIN de leur carte facilement. Ces fonctionnalités sont décrites dans le diagramme des cas d'utilisation suivant.

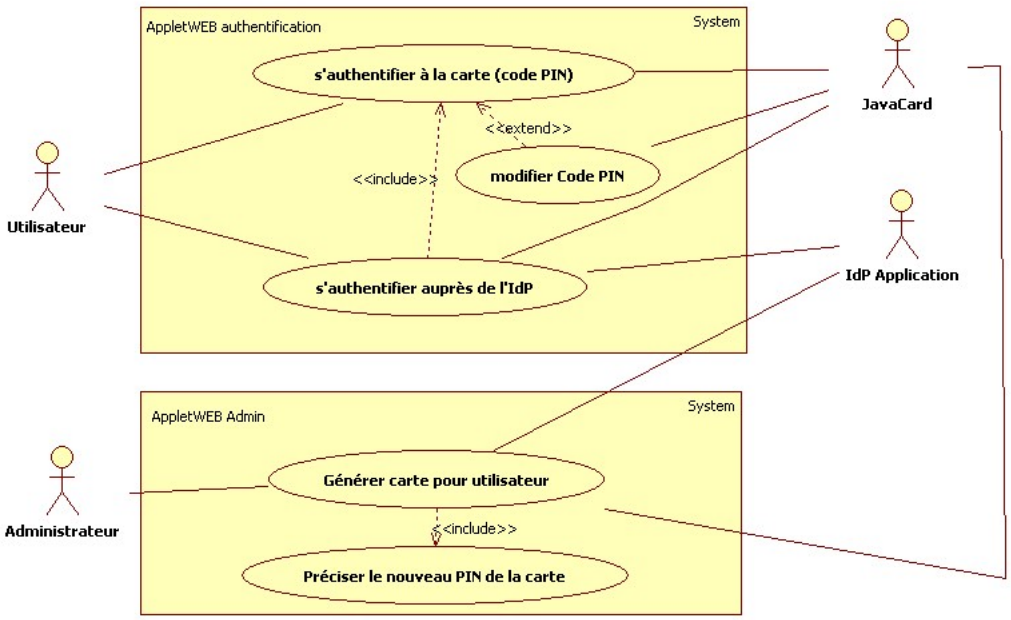

FIGURE 4.7 – Cas d'utilisations pour les applets

Voici les divers cas d'utilisations pour l'applet d'authentification dans le détail :

- S'authentifier à la carte (Code PIN) : ce cas d'utilisation permet d'ouvrir les fonctionnalités de la carte et donc par la même occasion de les protéger lorsque que l'utilisateur de la carte n'a pas le code PIN correspondant (comme pour les cartes de banque par exemple).
- Modifier le code PIN : il est bien entendu nécessaire pour qu'un code PIN soit efficace qu'il soit changé de temps en temps et connu seulement par le possesseur de la carte, c'est pour cela que cette fonction doit être offerte.
- S'authentifier auprès du fournisseur d'identité : ce cas d'utilisation est une procédure qui fait appel à des fonctions de la la carte à puce et du fournisseur d'identité. Ce processus sert essentiellement au transport du challenge et de sa réponse.

4.3.2 Communications (composants)

Les communications entre l'applet WEB et la carte et entre l'applet WEB et le fournisseur d'identité ont été abstraites dans des classe gérant elles-même la connexion et l'interfaçage avec leur entité respectives. Cela permet de cacher la complexité des différents transfert de paramètres et de valeurs de retour ainsi que différentes actions demandées pour un processus (transfert de clé, demande de challenge, etc).

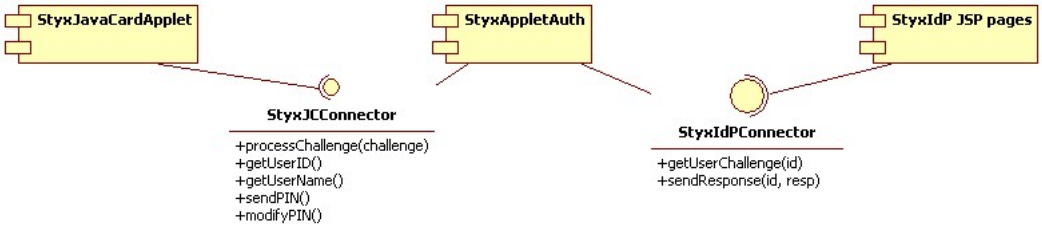

FIGURE 4.8 – Composants de communication, applet d'authentification

4.3.3 Séquence d'authentification

Le diagramme suivant montre la séquence d'authentification complète incluant l'authentification à la carte et l'authentification au fournisseur d'identité.

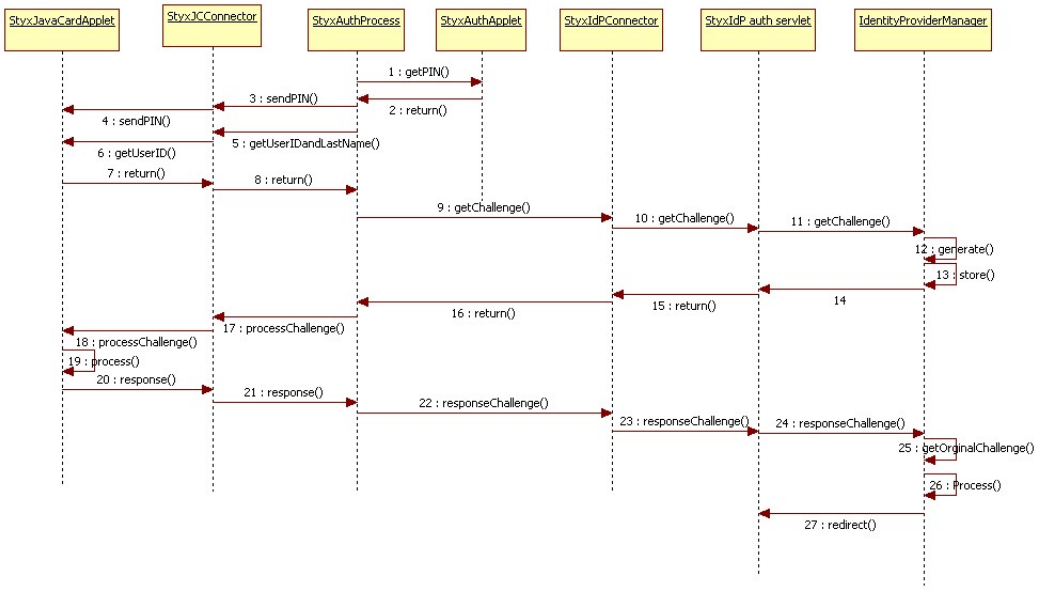

FIGURE 4.9 – Séquence authentification, applet d'authentification

Voici les principales actions requises lors de la procédure d'authentification :
- L'utilisateur doit d'abord s'identifier auprès de la carte pour pouvoir l'utiliser.
- Une fois authentifié, il récupère les paramètres lui correspondant à l'intérieur de la carte et les affiche.
- Ensuite il demande un challenge au serveur qui le crypte pour que seulement lui puisse le lire.
- La carte s'occupe de décrypter le challenge et de le re-crypter pour qu'il soit seulement lisible par le fournisseur d'identité
- Le fournisseur décrypte le challenge et valide ou non l'identification de l'utilisateur si le challenge reçu est égale au challenge généré.

4.3.4 Structure

La structure de l'applet WEB d'authentification est fournie dans le diagramme suivant. Mais voici une description générale des classes utilisées :
- StyxAppletAuth : GUI de l'applet.
- StyxAuthProcess : Processus d'authentification, à la carte et au fournisseur d'identité. Il inclut aussi le changement du code PIN. Pour plus d'informations sur ce processus, référez-vous au chapitre suivant.
- StyxContinueActionListener : Gestionnaire d'événement qui indique au processus StyxAuthProcess de continuer l'authentification avec le fournisseur d'identité.
- StyxIDPConnector : Interface de connexion au fournisseur d'identité.
- StyxJCConnector : Interface de connexion à la carte à puce.
- StyxModifyActionListener : Gestionnaire d'événement qui indique au processus StyxAuthProcess de continuer par la modification du code PIN.
- StyxStartActionListener : Gestionnaire d'événement qui indique au processus StyxAuthProcess de s'authentifier à la carte à puce.

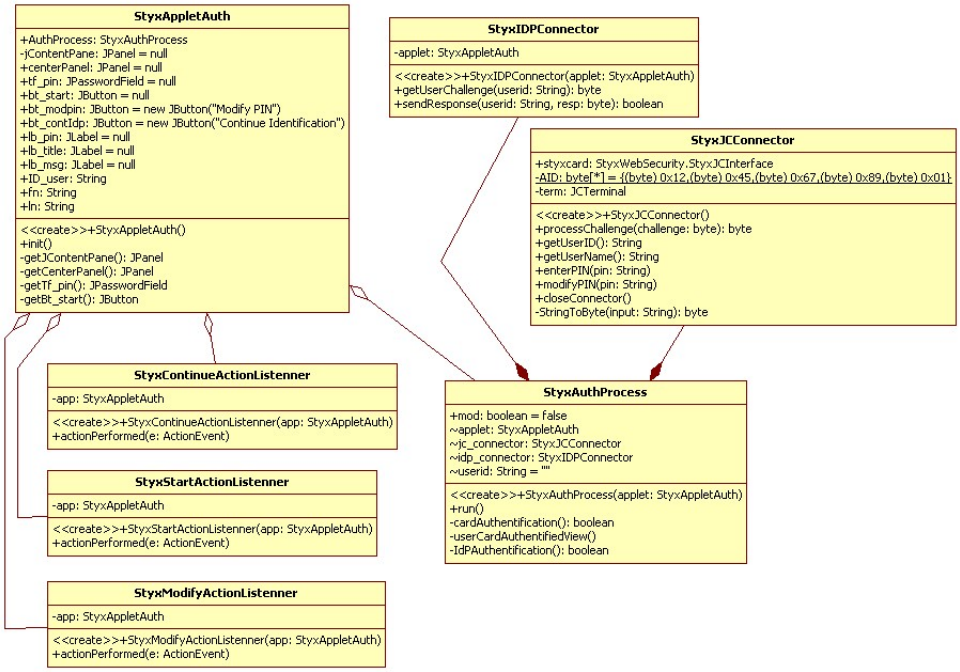

FIGURE 4.10 – Classes pour applet d'authentification

4.3.5 Processus d'authentification

Le diagramme d'état suivant nous montre les différents états dans lesquels le processus StyxAuthProcess peut se trouver lors de son utilisation.

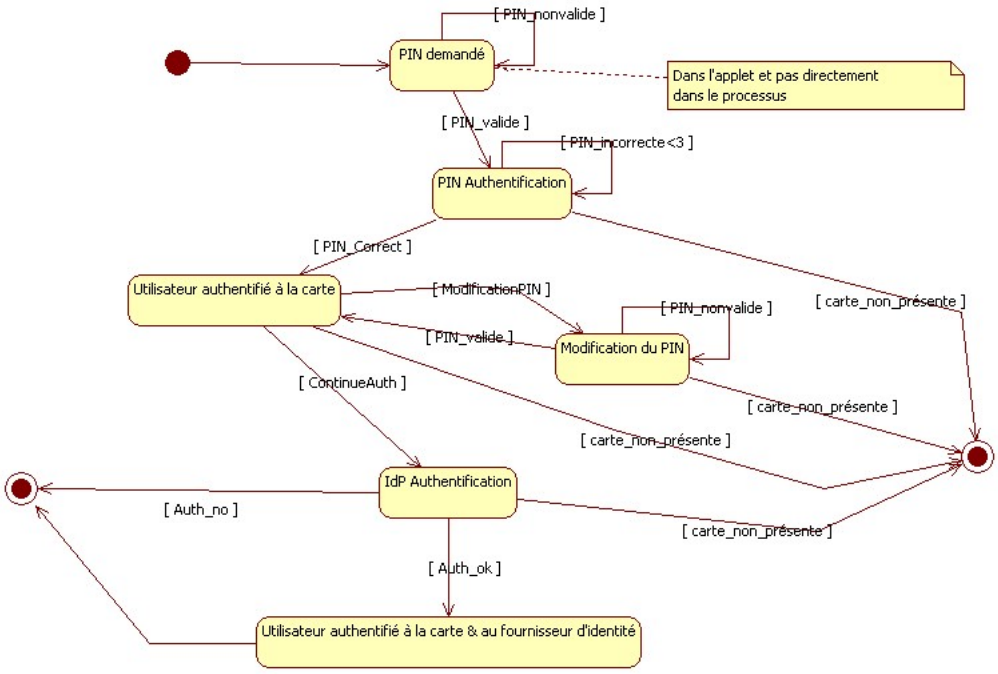

FIGURE 4.11 – Diagramme d'état pour le processus d'authentification

Ce diagramme d'état est nécessaire à la compréhension du processus. Car pour des raisons évidentes de sécurité une seule et unique connexion à la carte est initialisée. De se fait, le PIN n'est pas directement stocké dans l'applet après l'initialisation de la connexion avec la carte. Cela nous oblige donc à garder la connexion avec la carte ouverte durant toutes les étapes de l'authentification de l'utilisateur. A la fin de la séquence des états, la connexion n'est pas coupée avec la carte et le processus n'est pas suspendu. Ceci pour que le fournisseur d'identité s'assure que la carte est toujours présente dans le lecteur de l'utilisateur et aussi pour que l'échange de challenge/response continue.

4.4 Identity Provider

L'Identity Provider est le serveur qui fourni l'identité de l'utilisateur qu'il authentifie préalablement à des fournisseurs de service. Il comporte une partie permettant d'authentifier les utilisateurs, une autre permettant de communiquer de manière sécurisée avec les fournisseur de service et finalement une partie s'occupant de la gestion des utilisateurs, de leurs cartes et des Service Provider autorisés. Ce chapitre traitera principalement des fonctionnalités et de leurs modélisations.

4.4.1 Fonctionnalités

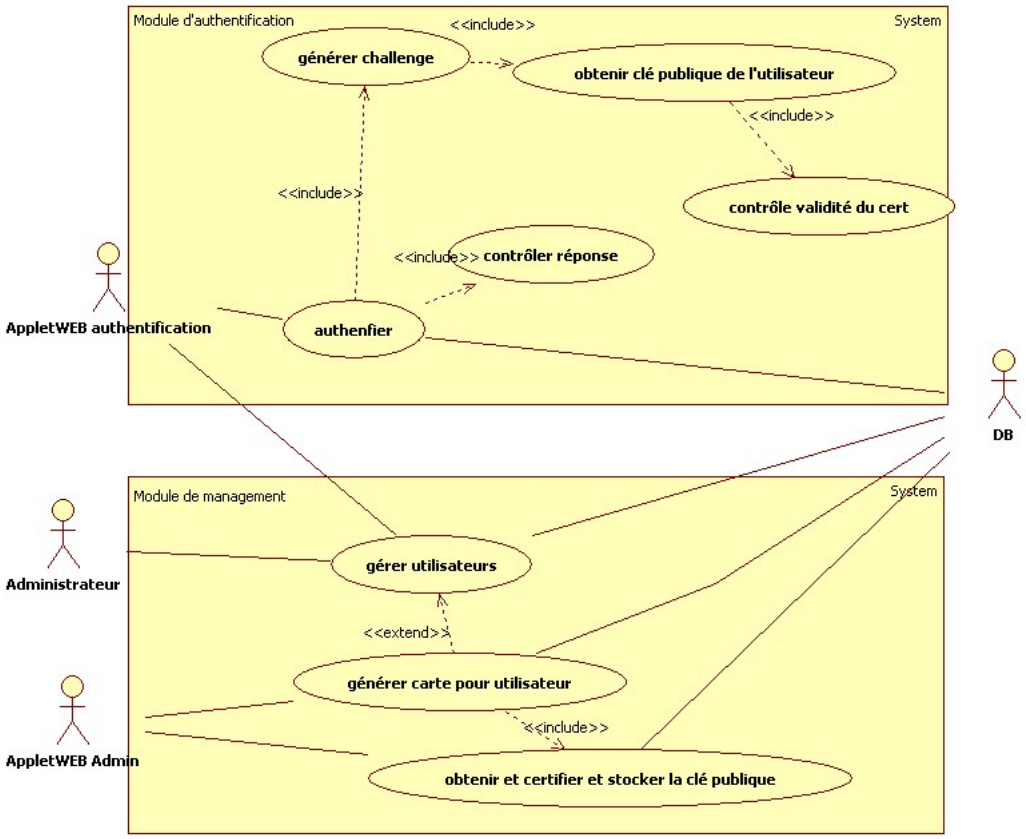

FIGURE 4.12 – Cas d'utilisations pour les applets

Voici d'abord en détail les fonctionnalités du module d'authentification :

- Authentifier : Le fournisseur d'identité doit être capable d'authentifier les utilisateurs de manière forte. Cela implique aussi une communication avec la carte à travers un applet WEB vu que le client doit être authentifié sur un site Internet.
- Générer un challenge : Pour authentifier un utilisateur la méthode challenge/response a été choisie. Cela implique la génération d'un challenge aléatoire. Ce challenge est

crypté et envoyé dans la carte de l'utilisateur qui le traitera pour le renvoyer au serveur d'identification.
- Contrôler la réponse : Pour authentifier l'utilisateur, il faut contrôler que le challenge envoyé précédemment soit bien traité et donc que la réponse soit correcte. C'est seulement après ce contrôle que l'utilisateur sera correctement authentifié.
- Obtenir la clé publique de l'utilisateur : Avant d'envoyer un challenge il faut le crypter avec la clé publique de l'utilisateur (dans ce cas seul l'utilisateur peut décrypter le challenge).
- Valider le certificat : La clé publique de l'utilisateur est stockée dans un certificat et doit être validée en contrôlant la chaîne de certification du certificat ainsi que sa période de validité.

Les fonctionnalités du module de gestion sont aussi importantes :
- Gérer les utilisateurs : On doit pourvoir ajouter/modifier/supprimer un utilisateur. On doit aussi pouvoir consulter son statut (sans carte/carte valide/carte expirée).
- Créer une carte pour un utilisateur : Créer une carte est une procédure complexe expliquée dans le sous-chapitre "Séquence de génération d'une carte", dans le chapitre "Applet WEB Administration". Voici cette procédure simplifiée :
  1. Installation de l'application dans la carte
  2. Modification du code PIN
  3. Inscription d'informations utilisateur (identifiant, nom, prénom)
  4. Génération d'un paire de clés RSA 2048bit
  5. Exportation de la clé publique hors de la carte
  6. Création d'un certificat avec cette clé et signature du certificat par le fournisseur d'identité
  7. Enregistrement de la publique du fournisseur d'identité dans la carte

  Toutes ces opérations se dérouleront à travers un applet WEB.
- Obtenir la clé publique de la carte et la certifier : Ce cas d'utilisation est décrit dans le précédent. C'est au fournisseur d'identité de s'en charger.
- Gérer les fournisseurs de service : Il faut pouvoir autoriser seulement certain fournisseur de service à demander une identification d'utilisateur. Les fournisseurs de service doivent donc être authentifiés avec le fournisseur d'identité. Cette gestion se fait par l'Identity Provider qui crée et fournit les paires de clés et certificats, mais aussi qui enregistre et certifie les fournisseurs de service autorisés.

4.4.2 Structure

Le fournisseur d'identité est avant tout une application web qui peut se diviser en plusieurs parties distinctes telles que : gestion, communication applet d'administration, communication applet d'authentification, authentification, et communication SAML.

Authentification Le module d'authentification comporte une seule page qui contient l'applet chargé d'accéder au lecteur de carte et de transmettre les challenge/réponse entre le fournisseur d'identité et la carte de l'utilisateur. Cette page est `login.jsp`.

Gestion Pour la gestion, plusieurs pages sont nécessaires. En voici un aperçu :
- `admin.jsp` Identification pour rentrer dans l'espace d'administration.
- `Management/provider_admin.jsp` Modification des options du serveur d'identification.

- `Management/sp_add.jsp` Ajout d'un Service Provider.
- `Management/sp_del.jsp` Confirmation de suppression et suppression d'un Service Provider.
- `Management/sp_view.jsp` Visualisation des Service Providers enregistrés.
- `Management/user_add.jsp` Ajout d'un utilisateur
- `Management/user_del.jsp` Confirmation de suppression et suppression d'un utilisateur.
- `Management/user_gsc.jsp` Création d'une carte pour un utilisateur.
- `Management/user_man.jsp` Visualisation des utilisateurs et de leurs statuts.
- `Management/user_mod.jsp` Modification d'un utilisateur.
- `Management/user_view.jsp` Visualisation des détails concernant un utilisateur.

Communication avec l'applet d'administration Les communications avec l'applet d'administration sont les suivantes et sont gérées par `StyxWebSecurity\Servlets\admin.java` :

- Créer et enregistrer un certificat pour l'utilisateur dans le fournisseur d'identité.
- Obtenir le certificat du fournisseur d'identité stocké dans la base de données.
- Obtenir les clés du CardManager stockées dans la base de données.

Communication avec l'applet d'authentification Les communications avec l'applet d'authentification sont les suivantes et sont gérées par `StyxWebSecurity\Servlets\auth.java` :

- Obtenir un challenge pour l'utilisateur qui le demande.
- Traiter la réponse envoyé par l'applet WEB à un challenge.
- Déconnecter l'utilisateur en cours.
- Savoir si le serveur à une demande de redirection vers un Service Provider en cours ou non, afin de connecter l'utilisateur.

Communication SAML Les communications avec les fournisseurs de service se font par l'intermédiaire des pages suivantes :

- `recv_saml_auth_request.jsp` Reçoit et contrôle la validité (signature du Service Provider) d'une requête d'authentification.
- `saml_auth.jsp` Répond à une requête d'authentification (après sa réception et son contrôle).

Voici plus en détail le processus de réponse à une requête d'authentification d'un fournisseur de service au fournisseur d'identité avec l'aide du graphique de flux suivant :

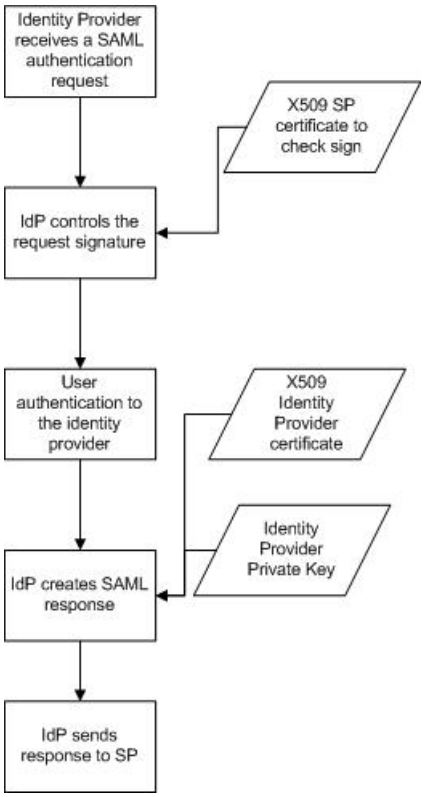
Identity Provider receives a SAML authentication request
X509 SP certificate to check sign
IdP controls the request signature
User authentication to the identity provider
X509 Identity Provider certificate
Identity Provider Private Key
IdP creates SAML response
IdP sends response to SP

FIGURE 4.13 – SAML, Communication du côté du fournisseur d'identité

1. Le fournisseur d'identité reçoit une requête d'authentification du fournisseur de service.
2. Il contrôle ensuite la signature de cette requête avec le certificat du fournisseur de service qu'il a dans sa base de donnée et qu'il a d'ailleurs aussi signé.
3. Il doit ensuite authentifier l'utilisateur, dans notre cas à l'aide de l'authentification forte et d'une carte à puce.
4. Après cette identification, le fournisseur d'identité doit créer une réponse d'authentification qui doit être signée avec sa clé privée et qui doit posséder son certificat publique.
5. Cette réponse est ensuite envoyée au fournisseur de service qui a fait la requête.

Utilisateur toujours authentifié D'une certaine manière l'utilisateur est toujours lié au fournisseur d'identité avec l'applet d'authentification. Mais il est aussi toujours lié au fournisseur d'identité depuis le fournisseur de service. Car si l'utilisateur se déconnecte du fournisseur d'identité, il doit se déconnecter du fournisseur de service. Hors pour que cette fonctionnalité soit possible, le fournisseur de service doit communiquer avec le fournisseur d'identité.
Il faudrait utiliser des Web services pour la communication entre les 2 fournisseurs. C'était impossible dans les délais impartis. Une solution de remplacement pour les besoins de la démonstration a donc été adoptée. Cette solution est hébergée sur l'Identity Provider et est

appelée et chargée à la connexion d'un utilisateur au fournisseur de service. Elle est composée de 2 pages :

- `login_control.jsp` Contient un java script qui fait des requêtes asynchrones au fournisseur d'identité pour savoir si l'utilisateur courant est toujours identifié. Sinon il déconnecte l'utilisateur du fournisseur de service.
- `check_auth.jsp` Fournit au Java script présent dans la page précédente les informations concernant la connexion de l'utilisateur au fournisseur d'identité.

La page `login_control.jsp` est chargé par le fournisseur de service, et est invisible à l'utilisateur (cadre). Elle appelle une page de déconnexion présente sur le fournisseur de service.

Logique métier La logique métier se trouve en partie dans le code des pages JSP mais aussi dans les classes présentent dans le package `StyxWebSecurity\IdP`.

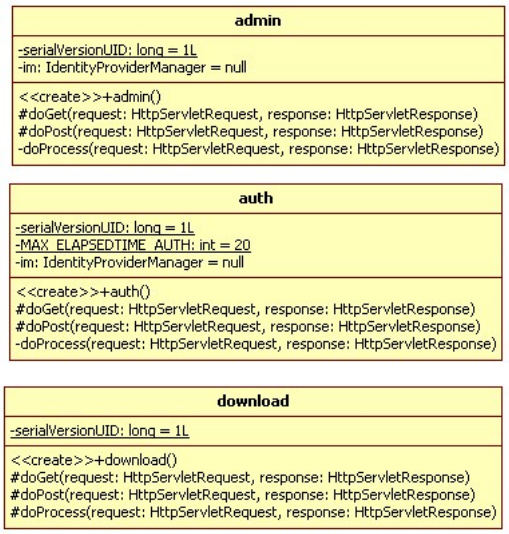

FIGURE 4.14 – Diagramme de classe des servlets

Les 3 servlets servent comme leurs noms l'indiquent à communiquer avec l'applet d'authentification, avec l'applet d'administration et à télécharger des fichiers qui sont présents en mémoire ou dans la base de données. Tous les servlets ont 3 méthodes, pour plus de commodité les requêtes arrivant dans les `doGet()` et `doPost()` sont transférées dans les `doProcess()` ou les classes de la logique métier sont utilisées pour traiter les différentes demandes.

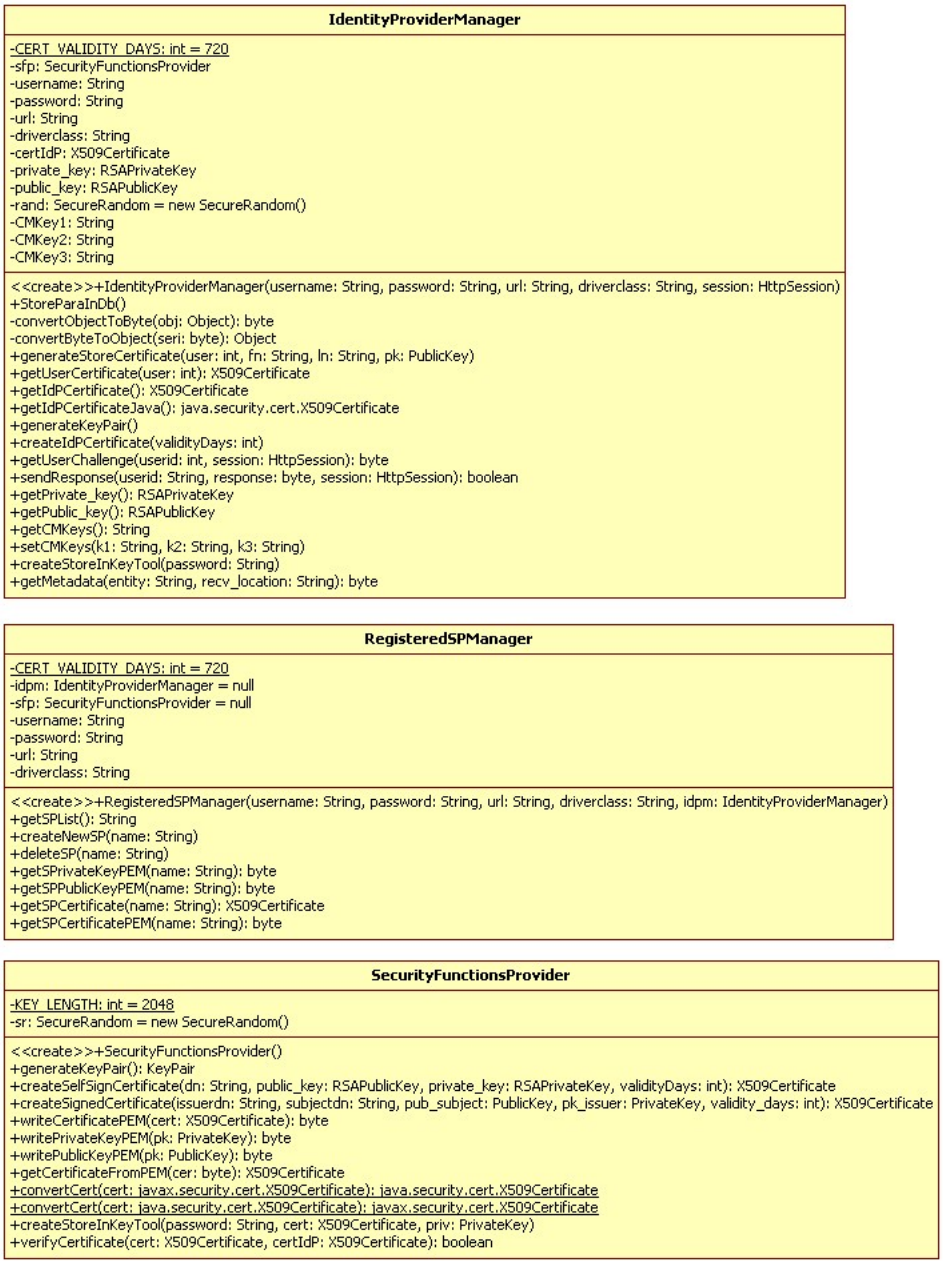

FIGURE 4.15 – Diagramme de classe de la logique métier de l'identity provider

La logique métier est séparée en trois grandes classes :

- `SecurityFunctionsProvider` Cette classe a toute les fonctions de cryptage nécessaires au traitement des différentes requêtes.
- `RegisteredSPManager` Cette classe contient toute la logique métier permettant de gérer

les Service Providers (création, clés, certificats, enregistrement dans la base)
- `IdentityProviderManager` Cette classe contient la logique métier relative aux paramètres du serveur et à la gestion des utilisateurs (création / suppression / visualisation / génération de carte).

4.5 Service Provider

Le fournisseur de service n'a pas beaucoup de fonctionnalités. La première est de fournir un service. La seconde, la plus importante est d'utiliser un fournisseur d'identité pour authentifier les utilisateurs de ce service. La logique métier est entièrement contenue dans les pages JSP à l'exception du toolkit websso et de toutes les librairies requises pour le faire fonctionner. Voici donc les pages nécessaires :

- `recv_saml.jsp` Reçoit et traite une réponse d'authentification envoyée par le fournisseur d'identité.
- `request_auth.jsp` Envoi une requête d'authentification au fournisseur d'identité.
- `install_keys.jsp` Permet d'installer les clés et certificats relatifs à l'utilisation de Tomcat en version sécurisée SSL.
- `protected.jsp` La page qui ne peut être accédée qu'après authentification.
- `login.jsp` Connexion de l'utilisateur si il n'est pas déjà authentifié au fournisseur de service. Dans ce cas il appelle `request_auth.jsp`.
- `logout.jsp` Déconnexion de l'utilisateur en cours du fournisseur de service.
- `index.html` Page de démonstration qui appelle `login.jsp` lorsque l'on veut se connecter à l'espace sécurisé.

Voici le processus simplifié de traitement des communications SAML entre le fournisseur de service et le fournisseur d'identification.

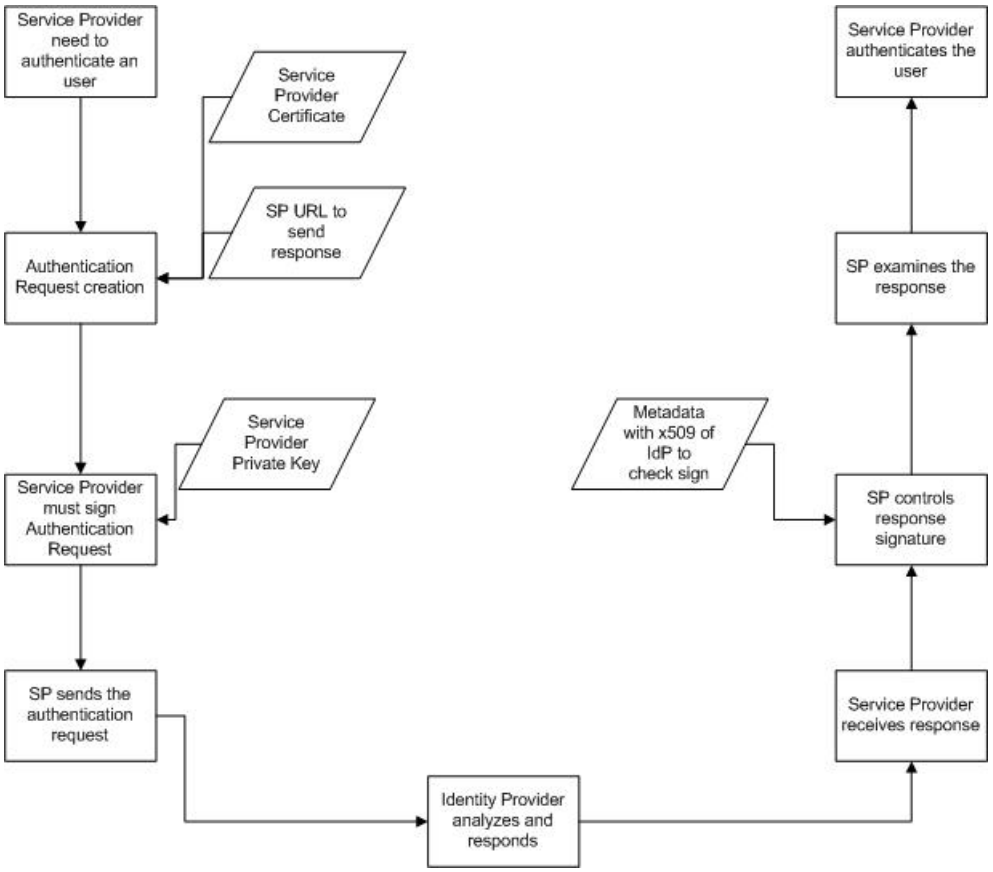
Service Provider need to authenticate an user
Service Provider Certificate
SP URL to send response
Authentication Request creation
Service Provider Private Key
Service Provider must sign Authentication Request
SP sends the authentication request
Identity Provider analyzes and responds
Service Provider receives response
Metadata with x509 of IdP to check sign
SP controls response signature
SP examines the response
Service Provider authenticates the user

FIGURE 4.16 – SAML, Processus de traitement du côté du fournisseur de service

1. Le fournisseur a besoin d'authentifier un utilisateur. Pour se faire il contacte un fournisseur d'authentification.
2. Le contact s'établit seulement après la création d'une requête d'authentification (document XML) qui contient principalement le certificat du fournisseur de service et l'adresse où le fournisseur d'identité doit lui soumettre la réponse.
3. Le fournisseur de service signe la requête en utilisant sa clé privée pour garantir la provenance des données.
4. La requête est ensuite transmise selon un des "bindings" prévu dans SAML, soit dans un HTTP POST soit dans un HTTP Redirect.
5. Le fournisseur d'identité traite la demande et envoi la réponse au fournisseur de service.
6. Le fournisseur de service reçoit la réponse.
7. Il contrôle la signature du document XML reçu avec le certificat du fournisseur d'identité qu'il télécharge dans un fichier de metadata.
8. Il examine ensuite la réponse.
9. Il authentifie ou non l'utilisateur pour le service.

4.6 Déploiement

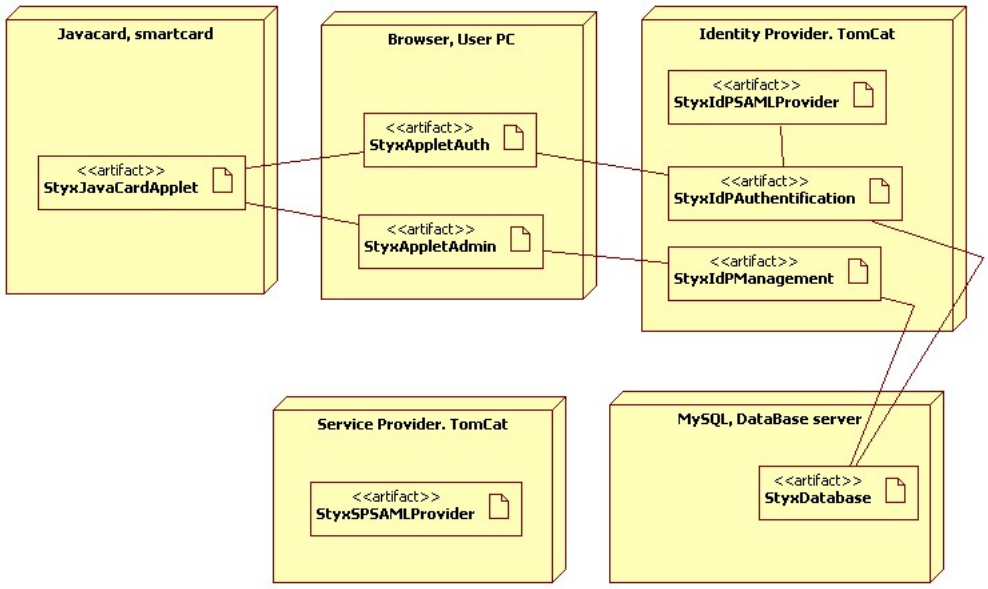

FIGURE 4.17 – Déploiement des composants

Le diagramme de déploiement ci-dessus montre les différentes plateformes utilisées pour créer le système décrit dans ce rapport. Les JavaCard sont utilisées du côté client ainsi que les applets WEB qui sont exécutés dans le navigateur Internet. Ils facilitent la connexion à la carte à travers le lecteur. L'Identity Provider et le Service Provider sont des serveurs d'application WEB Tomcat. Le serveur de base de donnée permet à l'Identity Provider d'enregistrer et de traiter les données relatives à la gestion des utilisateurs, des Service Provider et de ses paramètres.

Les composants présents dans l'Identity Provider sont construit avec de servlets, de différentes classes java et de différentes pages WEB. Voici le contenu de ces composants :

- StyxIdPSAML : `recv_saml_auth_request.jsp`,`saml_auth.jsp`, le toolkit de Clareity Security.
- StyxIdPAuthentication : `login.jsp`, le servlet `auth.java`, la classes du package `StyxWebSecurity.IdP`.
- StyxIdPManagement : toutes les pages présentes dans `Management`, les servlets `admin.java` et `download.java`, la classes du package `StyxWebSecurity.IdP`.

4.7 Base de données

La base de données est simple et comporte peu de données, voici les 3 tables utilisées :

- Users : stocke les utilisateurs, un tuple par utilisateur.
  - ID : auto increment number, primary key

  - First name, varchar
  - Last name, varchar
  - Address, varchar
  - Zip, varchar
  - Country, varchar
  - Phone, varchar
  - Email, varchar
  - PublicCert, blob (public certificate in PEM format)
- RegSP : stocke les fournisseurs de service enregistrés, un tuple par fournisseur de service.
  - Name : varchar, primary key
  - Private key : blob (private key in PEM format)
  - Public key : blob (public key in PEM format)
  - Certificate : blob (public certificate in PEM format)
- IdPSettings : Ne stocke qu'un tuple qui sont les paramètres du fournisseur d'identité.
  - Admin login : varchar
  - Admin password : varchar
  - Java Public Key : blob (serialized java object)
  - Java Private Key : blob (serialized java object)
  - Certificate : (public certificate in PEM format)
  - CardManager key 1 : varchar
  - CardManager key 2 : varchar
  - CardManager key 3 : varchar

Conclusion

La phase de spécification s'est bien déroulée. Elle a permis de faciliter la compréhension et même l'implémentation de certaines parties complexes de l'infrastructure, tels les applets web d'administration et d'authentification. La modélisation UML est outil puissant qui permet de vaincre la complexité à travers des schémas structurés. Elle est très utile même si elle n'est pas appliquée complètement à tout le projet.

5 Implémentation

Introduction

Ce chapitre a pour but d'exprimer les difficultés rencontrées lors de l'implémentation. Il permet aussi de mieux comprendre le fonctionnement global du prototype. Chaque élément de l'infrastructure est présent en sous-chapitre : Applet JavaCard, Applet WEB d'administration, Applet WEB d'authentification, Identity Provider et Service Provider. Pour chacun de ses éléments les points importants de leur codage sont développés.

5.1 Applet JavaCard

5.1.1 Contraintes

Il y a plusieurs contraintes importantes à noter. Elles nous empêchent de coder proprement et de spécifier simplement certaines fonctionnalités. La taille maximum des tableaux de bytes passé sur RMI lors d'un appel de méthode (que cela soit en tant que paramètre ou en temps que réponse) est de 254. Cela implique plusieurs méthodes pour envoyer ou récupérer des données sur la carte. Voici plus précisément les données posant problèmes :

- Clé publique : La clé RSA en 2048bits, qui a été choisie, a un modulo de 256 bytes. Si une clé en 1024bits moins sécurisée avait été choisie, ses problèmes auraient été évités car le modulo aurait été de 128 bytes.
- Challenge : L'algorithme RSA avec le padding PKCS1 a été choisi. Le challenge crypté prend de toute manière 256 bytes avec un clé de 2048bit. Cela aurait aussi été évité avec une clé de 1024bit, car il prendrait 128 bytes.
- Response : la réponse est aussi crypté et subit donc les mêmes contraintes que le challenge.

Avec cette contrainte, on voit donc apparaître plusieurs méthodes pour la même action, mais aussi des variables temporaires pour stocker les données entre les opérations.

Le stockage des certificats n'est pas facilité sur la carte. Aucune fonction de l'API permet de le traiter. La seule possibilité immédiate est de les enregistrer dans la mémoire de la carte, mais ce n'est d'aucune utilité car la carte n'a aucune fonctions pour les traiter. C'est pour cela qu'aucun certificat n'est stocké sur la carte.

Les JavaCard ne possèdent que très peut de types utilisables, les "string" ou "integer" sont stockés sous forme de tableau de bytes. Cette contrainte concerne l'identifiant utilisateur, son nom et son prénom.

Les JavaCard n'ont pas de "garbage collector" automatique, lorsqu'on utilise le mot-clé `new` dans une méthode, on s'attend à ce que les objets ayant des références internes ou non statiques soient supprimés à la sortie de la méthode. Mais le problème est que la JavaCard ne traite pas les objets comme cela. Un objet ou tableau créé avec `new` est créé dans la mémoire eprom et est persistant (n'est pas détruit quand sa référence n'existe plus). Si l'on ne s'occupe pas de ce problème, la mémoire de la carte sature rapidement et l'applet devient inutilisable. Il faut à ce moment désinstaller l'applet et son contexte pour pouvoir réutiliser la carte. Heureusement plusieurs solutions ce présentaient lors de la détection du problème :

- Utiliser des objets statiques instanciés une seule fois. Cela implique de refaire le design de l'application. Mais l'avantage est la rapidité car il n'y a pas de "garbage collector".
- Utiliser une sorte de "garbage collector" en invoquant la méthode dont le rôle est de supprimer les objets persistants qui ne sont plus référencés.
  `JCSystem.requestObjectDeletion()`
- Utiliser des objets "transient", les références sont stockées dans l'eprom, le contenu dans la ram. Le seul problème c'est que ces objets et leurs contenus ne sont effacés que lors d'une sélection d'applet ou d'un redémarrage de la carte.

Malgré la différence de performance, c'est la deuxième solution qui a été choisie car elle évitait une restructuration complète de l'application. Mais pour les futures utilisations de la plate-forme JavaCard il est recommander de réfléchir directement lors du design de l'application, surtout si la performance est cruciale, aux objets "static" ou "transient".

5.1.2 Stockage des informations utilisateurs

Plusieurs paramètres externes sont stockés dans la carte, certains dépendent de l'utilisateur de la carte, et un autre du fournisseur d'identité. Voici donc ces paramètres et la façon dont ils sont stockés et transférés dans la carte.

- Id, nom, et prénom de l'utilisateur.
  - Stockage interne : Sous la forme d'objets persistants (tableau de bytes car String indisponible).
    ```
    //User information
    private  byte[] user_id; //User id
    private  byte[] user_fn; //User FirstName
    private  byte[] user_ln;//User LastName
    ```
  - Envoi sur carte : (transformation de String en tableau de bytes)
    ```
    public void sendUserInfos(String Id, String firstName,
    String lastName)throws Exception{
    styxcard.sendUserInfos(Id.getBytes(),
    firstName.getBytes(), lastName.getBytes());
    }
    ```
  - Réception depuis carte : (transformation de tableau de bytes en String)
    ```
    public String getUserName() throws Exception{
    return ""
    +(new String(styxcard.getUserFirstName()))
    +" "
    +(new String(styxcard.getUserLastName()));
    }
    ```
- Clé publique du fournisseur d'identité.
  - Stockage interne : Dans un objet persistant RSAPublicKey, et aussi 2 tableaux pour la clé avant génération (modulo et exposant).
    ```
    private  RSAPublicKey IDPPublicKey; //IdP Public Key

    private  byte[] IDPpkmod = new byte[MODULO_2048_SIZE];
    private  byte[] IDPpkexp;
    ```
  - Envoi sur carte : à travers plusieurs méthodes, 2 pour le modulo et une pour l'exposant.
    ```
    public void  setIDPPublicKeyModulusFHalf(byte[] tmp);
    public void  setIDPPublicKeyModulusLHalf(byte[] tmp);
    public void  setIDPPublicKeyExponent(byte[] tmp);
    ```
    Mais du côté serveur il faut d'abord découper la clé en modulo et exposant :
    ```
    byte[] buffer = key.getModulus().toByteArray();
    byte[] exponent = key.getPublicExponent().toByteArray();
    ```
  - Réception depuis carte : On récupère d'abord le modulo et exposant et on re-génère la clé
    ```
    BigInteger modulus = new BigInteger(1, buffer_modulo);
    BigInteger exponent = new BigInteger(1, buffer_expo);
    ```

```
RSAPublicKeySpec rpks = new RSAPublicKeySpec(modulus, exponent);
KeyFactory kf = KeyFactory.getInstance("RSA");
return (RSAPublicKey)kf.generatePublic(rpks);
```

– Génération de la clé sur la carte : On crée une clé vide et on lui donne le modulo et l'exposant.

```
IDPPublicKey=(RSAPublicKey)KeyBuilder.buildKey(
  KeyBuilder.TYPE_RSA_PUBLIC, KeyBuilder.LENGTH_RSA_2048, false);
IDPPublicKey.setExponent(IDPpkexp,
  (short)0, (short)IDPpkexp.length);
IDPPublicKey.setModulus(IDPpkmod,
  (short)0, (short)256);
```

5.1.3 Génération de clé et traitement du challenge

Les clés sont générées à l'intérieur de la carte avec le théorème des restes Chinois. La paire de clé a une longueur de 2048 bits ce qui impose que la taille de son modulo soit de 256 bytes. Elle est entreposée à l'intérieur de la mémoire persistante de l'applet dans l'eprom de la carte et est référencée à l'aide d'une référence vers un objet "KeyPair". Cette technique fonctionne correctement vu que les objets sont persistants entre les connexion/déconnexion(alimentation) de la carte. Voici donc comment elle est créée :

```
private static final short   RSA_KEY_LENGTH=2048;
kp= new KeyPair(KeyPair.ALG_RSA_CRT,RSA_KEY_LENGTH);
kp.genKeyPair();
```

Le générateur utilisé comporte un problème qu'il faudrait résoudre manuellement en changeant l'exposant avant la génération car il est toujours de 65535.

Le traitement d'un challenge consiste à :

1. Recevoir le challenge :

   ```
   public void sendChallengeFHalf(byte[] tmp);
   public void sendChallengeLHalf(byte[] tmp);
   ```

   Deux méthodes sont employées car RMI sur la JavaCard ne peut pas sérialiser ou dé-sérialiser des tableaux de bytes de plus de 254 bytes.

2. Décrypter le challenge reçu avec sa clé privée. (présent dans "processChallenge()")

   ```
   //Decrypt challenge
   Cipher cipherde = Cipher.getInstance(Cipher.ALG_RSA_PKCS1, true);
   cipherde.init(kpriv, Cipher.MODE_DECRYPT);
   byte[]t = new byte[MODULO_2048_SIZE];
   short l=cipherde.doFinal(challenge, (short)0,
     (short)challenge.length, t, (short)0);
   byte[] decryted = new byte[l];
   Util.arrayCopy(t, (short)0, decryted, (short)0, l);
   ```

   L'algorithme de padding utilisé est le PKCS1 car il était le seul directement utilisable avec le côté off-card présente dans l'Identity Provider. Pour employer correctement cet algorithme et selon sa spécification dans l'API de la JavaCard, les challenge n'excède pas 256-11 bytes.

3. Crypter le challenge décrypté précédemment avec la clé publique de l'Identity Provider. (présent dans "processChallenge()")

```
//Encrypt the decrypted challenge
Cipher cipheren = Cipher.getInstance(Cipher.ALG_RSA_PKCS1, true);
cipheren.init(IDPPublicKey, Cipher.MODE_ENCRYPT);
response = new byte[MODULO_2048_SIZE];
cipheren.doFinal(decryted, (short)0,
  (short)decryted.length, response, (short)0);
```

4. Transmettre la réponse.

   ```
   public byte[] getResponseFHalf();
   public byte[] getResponseLHalf();
   ```

   La contrainte des "2 méthodes" est la même que pour recevoir le challenge.

Après chaque traitement de challenge, il faut vider la mémoire des objets persistants qui ne sont plus référencés et donc plus utilisés avec cette méthode :

```
public void cleanMemory(){
JCSystem.requestObjectDeletion();
}
```

5.1.4 Sécurisation des cartes et de l'application (applets WEB)

L'accès à l'applet est protégé par un code PIN qui est vérifié à chaque appel de fonction. Il est initialisé lors de l'installation de la carte avec une valeur choisie par l'administrateur. L'utilisateur peut le changer lors d'une de ces phases d'authentification. Déclaration du code PIN en attribut de l'applet :

```
private static OwnerPIN pin; //Applet PIN Code
```

Ensuite il est initialisé à l'installation de l'applet :

```
private static final byte PIN_TRYLIMIT=0x03;
private static final byte PIN_MAXSIZE=0x04;
private static final byte[] DEFAULT_PIN={(byte) 0x12,(byte) 0x34};
private static final byte DEFAULT_PIN_LENGTH=0x02;

pin = new OwnerPIN(PIN_TRYLIMIT,PIN_MAXSIZE);

pin.update(DEFAULT_PIN, (short)0, DEFAULT_PIN_LENGTH);
```

Avant d'utiliser l'applet, l'utilisateur doit entrer le pin et peut le faire grâce à cette méthode :

```
private static final byte DEFAULT_PIN_LENGTH=0x02;

pin.check(pin, (short)0, DEFAULT_PIN_LENGTH);
```

Pour sécuriser chaque méthode on insère comme première instruction de chaqunes d'elles l'instruction suivante :

```
if(pin.isValidated()==false)
  ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
```

Voici finalement la méthode pour le modifier :

```
public void modifyPIN(byte[] pin) throws RemoteException {
if(thi.pin.isValidated()==false)
  ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
if(pin.length!=2)return;//New PIN size error muste be 4 digits
this.pin.update(pin, (short)0, DEFAULT_PIN_LENGTH);
}
```

La cartes de type JavaCard possèdent toutes un gestionnaire de carte (CardManager) qui permet d'installer, de sélectionner, de supprimer des applets. Il permet aussi de configurer la sécurité de la carte. Pour accéder à ce gestionnaire il faut entrer les 3 clés nécessaires. Pour les cartes JCOP elles sont, à l'origine, toutes de cette valeur "404142434445464748494a4b4c4d4e4f". Dans un premier temps il faut se connecter à la carte et obtenir l'objet CardManager :

```
protected static byte[] JCOP_CARD_MANAGER_AID =
   { (byte) 0xa0, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00};

//Initialize connection to card manager
  JCTerminal terminal =JCTerminal.getInstance("pcsc:4", null);
terminal.open();
JCard card = new JCard(terminal, null, 100);
card.reset();
CardManager cardManager = new CardManager(card, JCOP_CARD_MANAGER_AID);
```

Ensuite il faut créer les clés et les soumettre au CardManager :

```
protected static byte[] JCOP_KEY1 = "@ABCDEFGHIJKLMNO".getBytes();
protected static byte[] JCOP_KEY2 = "@ABCDEFGHIJKLMNO".getBytes();
protected static byte[] JCOP_KEY3 = "@ABCDEFGHIJKLMNO".getBytes();

OPKey key4 = new OPKey(255, 1, OPKey.DES_ECB, JCOP_KEY1);
OPKey key5 = new OPKey(255, 2, OPKey.DES_ECB, JCOP_KEY2);
OPKey key6 = new OPKey(255, 3, OPKey.DES_ECB, JCOP_KEY3);

cardManager.setKey(key4);
cardManager.setKey(key5);
cardManager.setKey(key6);
cardManager.select();
cardManager.initializeUpdate(255, 0, CardManager.SCP_02_15);
cardManager.externalAuthenticate(OPApplet.APDU_ENC);
```

Une fois authentifié, on peut modifier les clés présentes :

```
cardManager.storeKeyset(Keys_STYX, 0x01);
```

Mais attention, on ne peut pas modifier les clés présentes en position 255, par contre s'il l'on ajoute un triplet de clés entre 1 et 254, le triplet présent en 255 n'est plus en service.

5.2 Applets Web Administration et Authentification

5.2.1 Interface de l'applet WEB d'administration

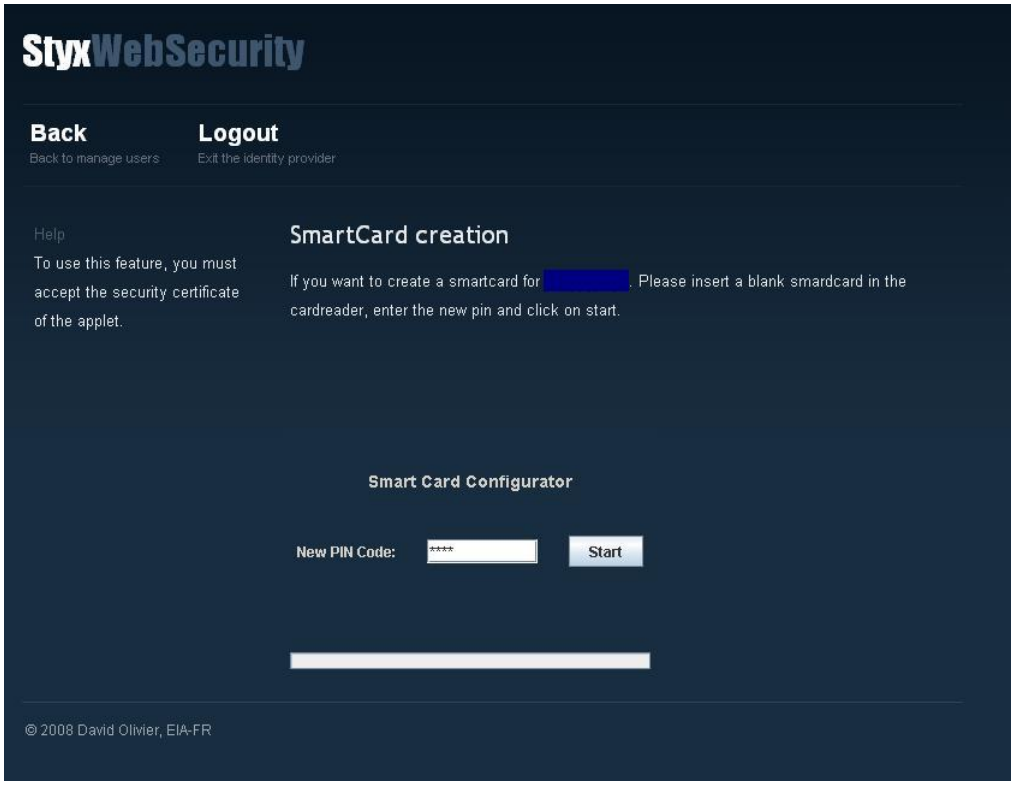
StyxWebSecurity
Back
Back to manage users
Logout
Exit the identity provider
Help
To use this feature, you must accept the security certificate of the applet.
SmartCard creation
If you want to create a smartcard for . Please insert a blank smardcard in the cardreader, enter the new pin and click on start.
Smart Card Configurator
New PIN Code:
****
Start
© 2008 David Olivier, EIA-FR

FIGURE 5.1 – Applet d'administration, implémentation, demande du pin

FIGURE 5.2 – Applet d'administration, implémentation, carte en cours de création

FIGURE 5.3 – Applet d'administration, implémentation, création de la carte terminée

5.2.2 Interface de l'applet WEB d'authentifcation

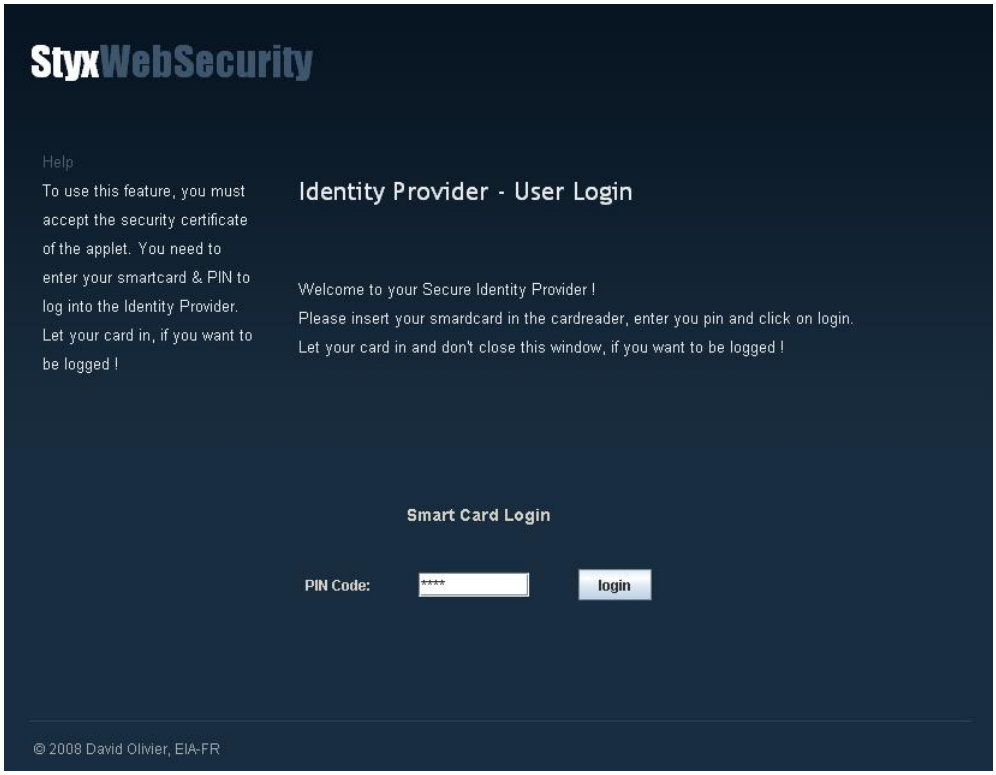
StyxWebSecurity
Help
To use this feature, you must accept the security certificate of the applet. You need to enter your smartcard & PIN to log into the Identity Provider. Let your card in, if you want to be logged !
Identity Provider - User Login
Welcome to your Secure Identity Provider !
Please insert your smardcard in the cardreader, enter you pin and click on login.
Let your card in and don't close this window, if you want to be logged !
Smart Card Login
PIN Code:
****
login
© 2008 David Olivier, EIA-FR

FIGURE 5.4 – Applet d'authentification, implémentation, demande du pin

FIGURE 5.5 – Applet d'authentification, implémentation, auhtentifié à la carte mais pas encore à l'Identity Provider

FIGURE 5.6 – Applet d'authentification, implémentation, authentifié à l'Identity Provider

FIGURE 5.7 – Applet d'authentification, implémentation, déconnecté à cause du retrait de la carte

5.2.3 Contraintes

- Installer le JRE 1.5 ainsi que ses plugins pour le navigateur car le pilote JCOP ne fonctionne pas avec les versions suivantes.
- Les applets doivent se trouver dans des archives jar signées pour avoir accès au pilote.

- Les applets doivent avoir le MANIFEST modifié pour accéder aux jars offcard.jar et rmioffcard.jar, qui sont aussi signés.
- Il faut que le fichier jct.dll se trouve dans le dossier `system32` de Windows sur la machine qui utilise les applets.
- Il faut accepter la signature des applets pour pouvoir les utiliser.
- Le connecteur de la carte doit toujours rester ouvert, sinon il faut ré-entrer le code PIN.
- Les communications à travers les firewalls sont utilisables seulement avec des requêtes GET ou POST sur le fournisseur d'identification.
- Les processus de traitement lourd doivent être séparés du processus de gestion de l'interface.

5.2.4 Communication avec la carte

La communication avec la carte dans les 2 applets WEB se fait par l'intermédiaire d'interface personnalisées et nommées "StyxJCInterface". Il faut bien entendu d'abord accéder au pilote du lecteur de carte "jct.dll" qui doit être installé sur le système. Dans cette interface on retrouve le code suivant qui permet d'accéder à l'objet distant (dans notre cas, c'est l'applet).

```
StyxWebSecurity.StyxJCInterface styxcard; //Remote object interface
JCTerminal term;             //Card Terminal object
JCard card;               //Java card object

//Open connection
term = JCTerminal.getInstance("pcsc:4", null);
term.open();
card = new JCard(term,null,5000);

//Get applet & distant object
JCApplet jca = new JCApplet(card,AID);
jca.select();
styxcard = (StyxWebSecurity.StyxJCInterface)
  RMIObjectFactory.getInitialReference(jca);
```

Maintenant que l'objet distant est accessible, la classe "StyxJCInterface" peut appeler toutes les méthodes définies dans l'interface suivante :

```
public interface StyxJCInterface extends Remote {

//Key methods
public void  generateKeyPair()  throws RemoteException;
public byte[] getPublicKeyModulusFHalf()  throws RemoteException;
public byte[] getPublicKeyModulusLHalf()  throws RemoteException;
public byte[] getPublicKeyExponent()  throws RemoteException;
public void  setIDPPublicKeyModulusFHalf(byte[] tmp)  throws RemoteException;
public void  setIDPPublicKeyModulusLHalf(byte[] tmp)  throws RemoteException;
public void  setIDPPublicKeyExponent(byte[] tmp)  throws RemoteException;
public void  generateIDPPublicKey()  throws RemoteException;

//Challenge methods
public void sendChallengeFHalf(byte[] tmp) throws RemoteException;
```

```
public void sendChallengeLHalf(byte[] tmp) throws RemoteException;
public void  processChallenge() throws RemoteException;
public byte[] getResponseFHalf()throws RemoteException;
public byte[] getResponseLHalf()throws RemoteException;

//User infos methods
public void sendUserInfos(byte[] id,byte[] fn,byte[] ln)
  throws RemoteException;
public byte[] getUserID()throws RemoteException;
public byte[] getUserFirstName()throws RemoteException;
public byte[] getUserLastName()throws RemoteException;

//PIN methods
public void   modifyPIN(byte[] pin) throws RemoteException;
public void   enterPIN(byte[] pin) throws RemoteException;
public void   cleanMemory() throws RemoteException;
}
```

5.2.5 Communication avec l'Identity Provider

La communication avec l'Identity Provider dans les 2 applets WEB se fait par l'intermédiaire d'interfaces personnalisées et nommées "StyxIDPInterface". Du côté du serveur on emploie 2 servlets, un pour l'applet d'administration et l'autre pour l'applet d'authentification. Les transferts de données (qui sont pour la plupart du temps des objets sérialisés) sont effectués par de requête HTTP de type GET ou POST. Cela permet d'éviter des problèmes de communication . Voici donc comment on récupère un objet sérialisé et envoyé depuis le serveur :

```
public RSAPublicKey getIdPPublicKey(){
try{
URL url = new URL(applet.getDocumentBase(),"admin?type=getIdPPK");
RSAPublicKey puk = (RSAPublicKey)
              (new ObjectInputStream(url.openStream())).readObject();
return puk;
}catch(Exception e){
System.out.println(e.toString());
return null;
}
}
```

On peut aussi transmettre des paramètres simples dans l'URL :

```
URL url = new URL(applet.getDocumentBase(),
   "admin?type=genStoCert&eid="+eid+"&efn="+efn+"&eln="+eln);
```

Et finalement on peut aussi transmettre des objets sérialisés au serveur à travers une requête POST :

```
public void generateStoreCertificate(String user_id,
                      String fn, String ln, RSAPublicKey pk){
try{
ObjectOutputStream out;
```

```
BufferedReader reader;
HttpURLConnection cnx;

//base settings
String eid = URLEncoder.encode(user_id,"ISO-8859-1");
String efn = URLEncoder.encode(fn,"ISO-8859-1");
String eln = URLEncoder.encode(ln,"ISO-8859-1");
URL url = new URL(applet.getDocumentBase(),
        "admin?type=genStoCert&eid="+eid+"&efn="+efn+"&eln="+eln);
cnx = (HttpURLConnection) url.openConnection();
cnx.setRequestMethod("POST");
cnx.setDoInput(true);
cnx.setDoOutput(true);
cnx.setUseCaches(false);
cnx.setDefaultUseCaches(false);
cnx.setRequestProperty("Content-Type", "application/octet-stream");
cnx.connect();
out = new ObjectOutputStream(cnx.getOutputStream());
out.writeObject(pk);
reader = new BufferedReader(
     new InputStreamReader(cnx.getInputStream()));
String ligne;
while ((ligne = reader.readLine()) != null) {
System.out.println(ligne);
}
out.flush();
out.close();
reader.close();
cnx.disconnect();
}catch(Exception e){
System.out.println(e.toString());
}
}
```

Les servlets sont décrits plus en détails dans le chapitre concernant l'Identity Provider.

5.2.6 Transfert du pilote jct.dll et du fichier StyxWebSecurity.cap

Les 2 fichiers sont transférés sur le poste qui possède le lecteur de carte. Le fichier CAP est seulement transféré lors de l'utilisation de l'applet d'administration car il contient l'application qui doit être installée sur la carte. Pour les transférer sur le poste client l'applet doit être signé et contenir ses fichiers dans son archive jar. Ensuite on utilise la méthode suivante :

```
public void copyResourceFromJarToBinDir(String file)
{
InputStream fileInputStream =
  getClass().getClassLoader().getResourceAsStream(file);
String javaHome = System.getProperty("java.home");
String fileSeparator = ""+System.getProperty("file.separator").charAt(0);
File outFile = null;
   try
```

```
   {
      BufferedInputStream  binStream  =
        new BufferedInputStream(fileInputStream);
      outFile = new File(javaHome+fileSeparator+file);

      BufferedOutputStream boutStream =
        new BufferedOutputStream(new FileOutputStream(outFile));
      int b = -1;
      while((b = binStream.read()) != -1)
      {
         boutStream.write(b);
      }
      binStream.close();
      binStream.close();
      boutStream.flush();
      boutStream.close();
   }
   catch(Exception e)
   {
       e.printStackTrace();
   }
}
```

Elle télécharge le fichier dans le dossier `bin` de la machine virtuelle Java utilisée par l'applet. Une autre méthode presque semblable copie le fichier soumis vers le dossier `System32` de Windows. Il est aussi envisageable de l'utiliser pour copier un pilote Linux ou tout autre fichier sur le poste client. Attention elle ne marche pas avec Windows Vista, car les répertoires systèmes sont protégés en écriture.

5.2.7 Installation de StyxWebSecurity.cap

Pour initialiser une carte vierge, il faut copier l'application sur la carte et l'installer. Sur la plateforme JavaCard c'est le CardManager qui s'en occupe. Il faut donc s'y enregistrer avant de commencer la copie :

```
CapFile cap = new CapFile(javaHome+fileSeparator+"StyxWebSecurity.cap", null);

byte[] cardManagerAid = cardManager.getAID();

cardManager.installForLoad(cap.pkgId, 0, cap.pkgId.length, cardManagerAid, 0,
cardManagerAid.length, null, 0, null, 0, 0, null, 0);

cardManager.load(cap, null, CardManager.LOAD_ALL,
null, cardManager.getMaxBlockLen());
```

Ensuite il faut rendre l'applet sélectionnable pour y accéder :

```
  byte[] capaid = cap.aids[0];
cardManager.installForInstallAndMakeSelectable(cap.pkgId, 0, cap.pkgId.length,
capaid,0, capaid.length, capaid, 0, capaid.length, 0, defaultInstallParam, 0,
defaultInstallParam.length, null, 0);
```

Après ces opérations, l'applet est installé et accessible dans la carte. On peut communiquer avec lui facilement en utilisant la technologie RMI.

5.2.8 Traitement et processus

Pour des questions relatives à la sécurité principalement, la connexion à la JavaCard ne doit pas être coupée durant tout le traitement. Sinon le code PIN devrait être stocké dans la partie applicative ou être re-demandé à l'utilisateur ce qui n'est pas envisageable. C'est d'une part pour cela que tout le traitement ce fait dans un processus. Ensuite il y a aussi la question de l'interface graphique qui est paralysée si des actions demandant des ressources sont traitées dans le même processus que l'affichage. C'est donc la deuxième raison qui nous oblige à utiliser un processus de traitement. Ceci est valable pour les 2 applets : "StyxNewCardProcess" et "StyxAuthProcess". Ces 2 classes implémentent donc Thread.

Il est nécessaire pour certaines fonctionnalités des applications de mettre les processus en pause à certain moment de leur traitement. Hors maintenant, les méthodes simples qui étaient "suspend()" et "resume()" sont dépréciées. Il faut donc utiliser d'autres fonctionnalités de Java pour mettre en pause les threads :

```
synchronized(this) {
 wait();
}
```

Et ensuite pour les relancer (exemple concernant le processus d'authentification) :

```
synchronized(app.AuthProcess) {
 app.AuthProcess.notify();
}
```

5.3 Identity Provider

5.3.1 L'interface

Voici l'interface du fournisseur d'identité, les figures suivantes montrent ses principales pages.

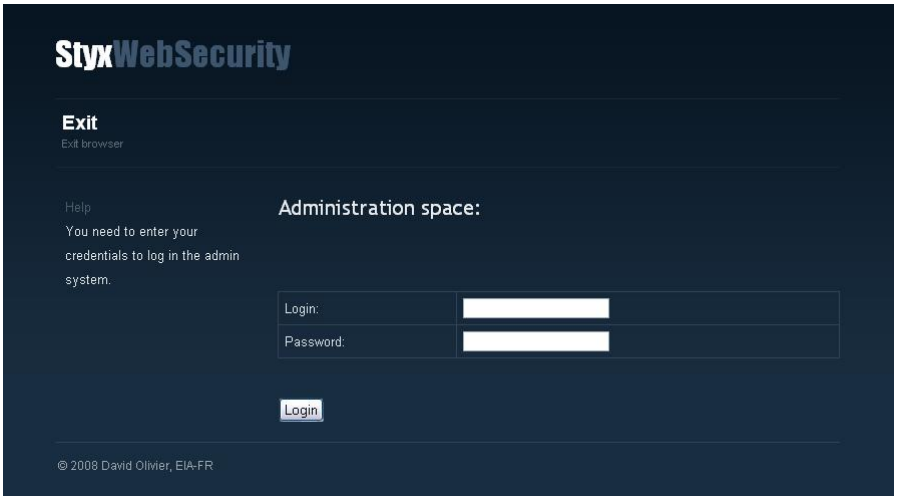

FIGURE 5.8 – Identity Provider, implémentation, page d'accès à l'espace d'administration

FIGURE 5.9 – Identity Provider, implémentation, page principale de l'espace d'administration

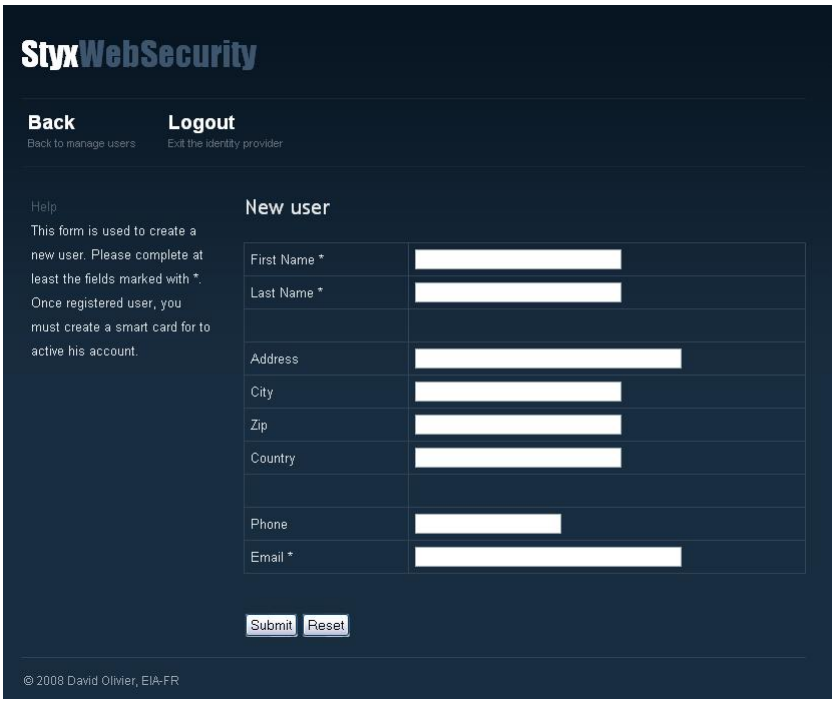

FIGURE 5.10 – Identity Provider, implémentation, page d'ajout de nouvel utilisateur

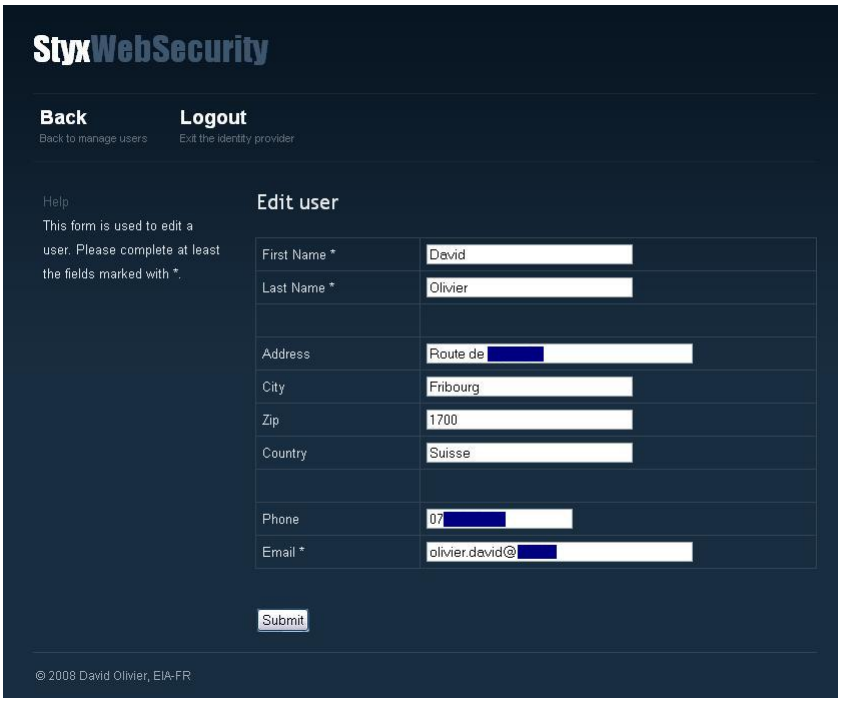

FIGURE 5.11 – Identity Provider, implémentation, page de modification d'un utilisateur

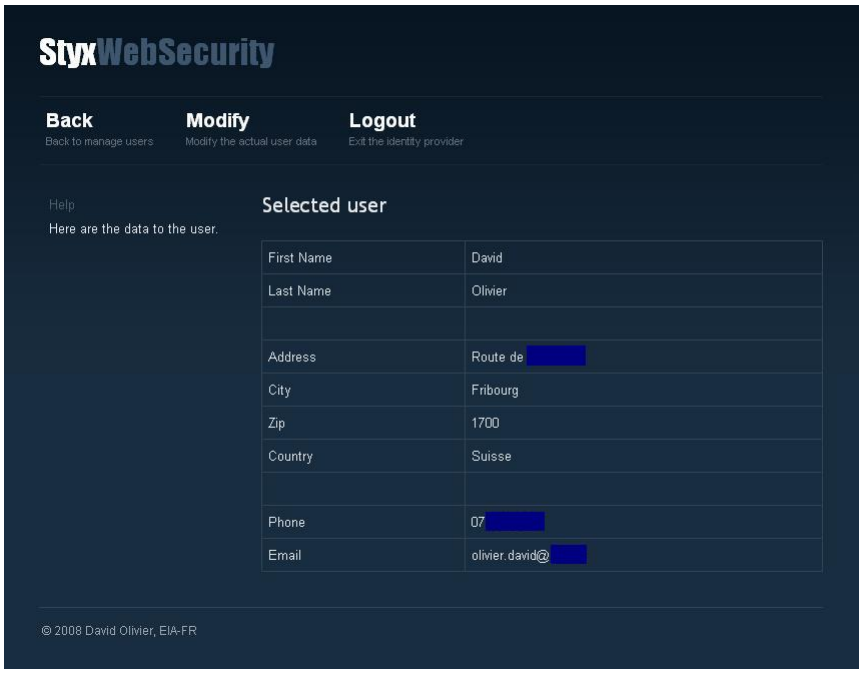

FIGURE 5.12 – Identity Provider, implémentation, page de visualisation d'utilisateur

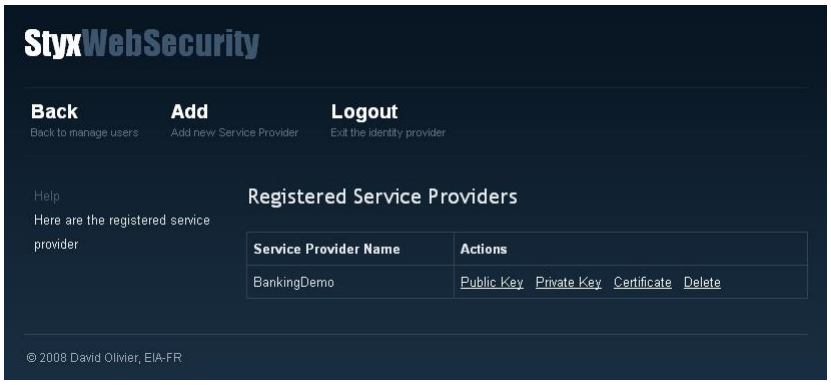

FIGURE 5.13 – Identity Provider, implémentation, page de visualisation des Service Providers

5.3.2 Les servlets d'authentification et d'administration

Fonctions du servlet d'authentification Voici ses principales fonctions :

- Fournir des challenges cryptés avec la clé publique de l'utilisateur qui les demande et les garder en mémoire. Les challenges sont créés avec un SecureRandom et garder en mémoire dans la session. Ils sont cryptés avec la clé publique présente dans le certificat

de l'utilisateur correspondant en utilisant un algorithme semblable à celui présent dans la carte.

```
SecureRandom rand = new SecureRandom();
rand.setSeed(System.nanoTime());

-----------------------------------

byte[] clear_challenge = new byte [128];
rand.nextBytes(clear_challenge);

-----------------------------------

Cipher cipher = Cipher.getInstance("RSA/None/PKCS1Padding");
X509Certificate usercert = getUserCertificate(userid);
cipher.init(Cipher.ENCRYPT_MODE, usercert.getPublicKey());
byte[] crypted = cipher.doFinal(clear_challenge);

-----------------------------------

session.setAttribute("challenge", clear_challenge);
session.setAttribute("user", Integer.toString(userid));
```

– Traiter une réponse à un challenge en le décryptant et le comparant à celui en mémoire. Il faut garder en mémoire que l'utilisateur est authentifié pendant une durée maximale de 20 seconde. Voilà comment on récupère ce challenge :

```
byte[] s_challenge = (byte[])session.getAttribute("challenge");
```

Il est ensuite comparé au challenge décrypté avec la clé privée du fournisseur d'identité :

```
Cipher cipher = Cipher.getInstance("RSA/None/PKCS1Padding");
cipher.init(Cipher.DECRYPT_MODE, private_key);
byte[] clear = cipher.doFinal(response);
if(Arrays.equals(clear, s_challenge)){...
```

Finalement on indique à la session que l'utilisateur est authentifié. Mais cela n'est permanent pour des raisons évidentes de sécurité. Si un utilisateur retire la carte et que l'ordre de déconnexion envoyé par l'applet n'arrive jamais au serveur, l'authentification sera de toute manière expirée, car il n'aura pas reçu de challenge depuis plus de 20 secondes.

```
private static final int MAX_ELAPSEDTIME_AUTH= 20; //in sec

-----------------------------------

request.getSession().setAttribute("authenticated", eid);
request.getSession().setMaxInactiveInterval(MAX_ELAPSEDTIME_AUTH);
```

– Supprimer la session de l'utilisateur immédiatement après le retrait de la carte. Pour

cette fonctionnalité, on utilise l'invalidation de session :

```
request.getSession().invalidate();
```

– Il faut contrôler la validité du certificat correspondant à l'utilisateur. Il est présent dans la base de données de l'Identity Provider. Le contrôle s'effectue par rapport au certificat public du fournisseur d'identité qui a signé le certificat, et par rapport à la date courante. La méthode suivante est appelée avant chaque utilisation d'un certificat d'utilisateur.

```
public boolean verifyCertificate(X509Certificate cert){
try{
cert.checkValidity();
cert.verify(certIdP.getPublicKey());
return true;
}catch(Exception e){
return false;
}
}
```

Fonctions du servlet d'administration Voici ses principales fonctions :

– Générer un certificat avec la clé publique de l'utilisateur et ses informations personnelles. Cela inclut la signature de ce certificat avec la clé privée de l'Identity Provider, et son stockage dans une base de donnée.
– Distribuer les clés du CardManager stockées dans la base de données. Afin que l'applet d'administration puisse installer le fichier CAP.
– Donner la clé publique du fournisseur d'identité pour son stockage dans la carte.

Envoi d'objet vers un applet WEB Il est très simple d'envoyer un objet sérialisé depuis le servlet vers l'applet car on peut utiliser le flux de sortie de la HttpServletResponse. Voici un exemple du transfert de l'objet RSAPublicKey :

```
ObjectOutputStream answer = new ObjectOutputStream(response.getOutputStream());
answer.writeObject(im.getIdPPublicKey());
answer.flush();
answer.close();
```

Réception d'un objet depuis un applet WEB La réception d'objet à travers un servlet est aussi commode que l'envoi. il faut employer le flux d'entré de la HttpServletRequest. Voici un exemple lorsque l'on reçoit la clé publique de l'utilisateur de la carte :

```
InputStream is=null;
ObjectInputStream ois=null;
is = request.getInputStream();
ois = new ObjectInputStream(is);
RSAPublicKey pk = (RSAPublicKey) ois.readObject();
```

Réception des paramètres transmis dans l'URL Il est très simple de récupérer ces paramètres :

```
String nompara = request.getParameter("nompara");
```

5.3.3 Logique métier

La logique métier se trouve dans les différentes pages JSP mais aussi dans les classes qui sont décrites ci-dessous :

- `SecurityFunctionsProvider` Cette classe a toute les fonctions de cryptage nécessaires au traitement des différentes requêtes. Elle utilise la librairie "BouncyCastle".
- `RegisteredSPManager` Cette classe permet de gérer les Service Providers (création, clés, certificats, enregistrement dans la base)
- `IdentityProviderManager` Cette classe permet de gérer les paramètres du serveur et à la gestion des utilisateurs (création/suppression/visualisation/génération de carte).

5.3.4 Signature des applets WEB

Les applets WEB qui servent soit à l'authentification "StyxWebSecurityAuthApplet.jar", soit à l'administration "StyxWebSecurityAdminApplet.jar", doivent être signés pour accéder au système de fichier du PC client et pour utiliser le pilote du lecteur de carte. Depuis la dernière version de Java 5.0, il n'existe plus qu'une manière pour signer les fichiers dans lesquels les applets mentionnés précédemment et les librairies d'accès à la JavaCard ("offcard.jar" et "rmioffcard.jar") sont contenus. Il faut employer le keytool de Java pour enregistrer les clés servant à la signature dans un keystore. Ensuite il faut signer les jars avec jarsigner.

Pour plus de sécurité il faut pouvoir signer les fichiers jars et les contrôler avec la clé privée et le certificat du fournisseur d'identité. Une fonction à été développée en employant le keytool pour les enregistrer et pour finalement les employer avec le jarsigner :

```
public void createStoreInKeyTool(String password){
try{
String keystorename = System.getProperty("keystore");

if (keystorename == null)
keystorename = System.getProperty("user.home")+
System.getProperty("file.separator")+
".keystore"; // especially this ;-)

KeyStore ks = KeyStore.getInstance("JKS", "SUN");
ks.load( null , password.toCharArray());

ks.store(new FileOutputStream ( keystorename  ),
password.toCharArray());
ks.load(new FileInputStream ( keystorename ),
password.toCharArray());

java.security.cert.Certificate[] certs =
```

```
  new java.security.cert.Certificate[1];
certs[0]=
  (java.security.cert.Certificate)getIdPCertificateJava();

ks.setKeyEntry("StyxIdP", (Key)getPrivate_key(),
password.toCharArray(),
certs );

ks.store(new FileOutputStream ( keystorename ),
password.toCharArray());
}catch(Exception e){
e.printStackTrace();
}
}
```

Avant la version 5.0 de Java, Il était possible d'utiliser la classe "sun.security.util.SignatureFile" qui n'est plus disponible pour signer directement depuis le code java des fichiers jar. Plusieurs exemples sont présents sur Internet, mais le plus pertinent reste disponible à cette adresse : http://www.onjava.com/pub/a/onjava/2001/04/12/signing_jar.html?page=1

Signature des jars il faut signer les fichiers avec ces commandes, mais attention il faut avoir dans le `path`, le dossier `bin` du JDK.

```
jarsigner StyxWebSecurityAdminApplet.jar styxidp
jarsigner StyxWebSecurityAuthApplet.jar styxidp
jarsigner offcard.jar styxidp
jarsigner rmioffcard.jar styxidp
```

5.3.5 SAML

Réception de requête d'identification La requête est réceptionné par la page JSP `recv_saml_auth_request.jsp`. Le framework SSO de Clareity Security est utilisé pour faciliter le traitement du SSO. Il est basé sur OpenSAML.

```
HttpHandler httpPost = new HttpHandler();
AuthnRequest authnRequest = httpPost.decodeSAMLRequest(request);
String relayState = httpPost.getRelayState();
```

Ensuite on extrait le document XML de la requête :

```
String messageXML = httpPost.getXMLSAMLRequest();
messageXML = messageXML.replace("<", "&lt;");
messageXML = messageXML.replace(">", "&gt;");
```

Contrôle des fournisseurs de service Le contrôle de l'expéditeur de la requête d'authentification se fait aussi dans la page JSP `recv_saml_auth_request.jsp`. Dans un premier on regarde l'émetteur de la requête et on va chercher son certificat dans la base de données :

```
Issuer is = authnRequest.getIssuer();
X509Certificate cert = rspm.getSPCertificate(is.getValue());
```

Ensuite on contrôle la validité du certificat et on crée le "validateur" de signature :

```
org.opensaml.xml.security.x509.BasicX509Credential
  publicCredential =
    new org.opensaml.xml.security.x509.BasicX509Credential();

if(!spf.verifyCertificate(cert,idpm.getIdPCertificate()))
  throw new Exception("cert not valid");
publicCredential.setPublicKey(cert.getPublicKey());

SignatureValidator signvalid = new
  org.opensaml.xml.signature.SignatureValidator(publicCredential);
```

Et finalement on contrôle la signature de la requête :

```
try{
  signvalid.validate(authnRequest.getSignature());
}catch(Exception e){
  out.println("Authentication request, signature not valid");
  return;
}
```

Envoi de réponse d'identification La réponse à la requête d'identification se déroule dans la page `saml_auth.jsp`. Dans un premier temps, il faut obtenir la clé publique et la clé privée du fournisseur d'identité pour signer la requête. Les classes `PublicKeyCache` et `PrivateKeyCache` sont utilisées pour obtenir et stocker les clés dans le cache de l'application. Ensuite si l'utilisateur est authentifié au fournisseur d'identité on crée la réponse :

```
SAMLResponse rsp =
  new net.clareitysecurity.websso.idp.SAMLResponse();
rsp.setAuthnRequest(auth);
rsp.setLoginId(authenticated);
String issuerName =
  application.getInitParameter("IdPName");
rsp.setIssuerName( issuerName );
rsp.setPrivateKeyCache(pkCache);
rsp.setPublicKeyCache(pubCache);
rsp.setSignAssertion(true);

String samlResponse = rsp.createSuccessResponse();
String responseXML = rsp.getResponseXML();
String actionURL = rsp.getActionURL();
responseXML = responseXML.replace("<", "&lt;");
responseXML = responseXML.replace(">", "&gt;");
```

Cette réponse est ensuite envoyée par un HTTP Post et du Java script pour automatiser :

```
<body onload="document.forms[0].submit()" >
<noscript>Your browser does not support
Javascript or you have disabled. Please
press the continue button to proceed.
</noscript>
<form
  action="<% out.print(actionURL); %>"
```

```
  method="post">
<input type="hidden" name="RelayState"
   value="<% out.print(relayState); %>">
<input type="hidden" name="SAMLResponse"
   value="<% out.print(samlResponse); %>">
<noscript>
  <input type="submit" value="Continue">
</noscript>
</form>
</body>
```

Fichier de metadata Le fichier de metadata est téléchargé par le fournisseur de service pour vérifier la signature de la réponse d'authentification. Il est proposé au téléchargement par le servlet `download.java`. Il est créé avec le certificat courant du fournisseur d'identité et son nom dans la classe `IdentityProviderManager` avec la méthode `getMetadata()`. Voici un exemple de ce fichier :

```
<?xml version="1.0" encoding="UTF-8" ?>
<md:EntityDescriptor
  xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
  entityID="ENTITYNAME">

<md:IDPSSODescriptor WantAuthnRequestsSigned="false"
  protocolSupportEnumeration
    ="urn:oasis:names:tc:SAML:2.0:protocol">

<md:KeyDescriptor use="signing">
  <ds:KeyInfo xmlns:ds=\"http://www.w3.org/2000/09/xmldsig#\">
    <ds:X509Data>
      <ds:X509Certificate>
      HERECERTIFICATEBASE64
      </ds:X509Certificate>
    </ds:X509Data>
  </ds:KeyInfo>
</md:KeyDescriptor>

<md:NameIDFormat>
urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
</md:NameIDFormat>

<md:NameIDFormat>
urn:oasis:names:tc:SAML:2.0:nameid-format:transient
</md:NameIDFormat>

<md:SingleSignOnService
  Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
  Location="URLSAMLRECVONIDP" />

</md:IDPSSODescriptor>
```

```
</md:EntityDescriptor>
```

5.4 Service Provider

5.4.1 Interface

Les figures suivantes montrent les principales pages du fournisseur de service.

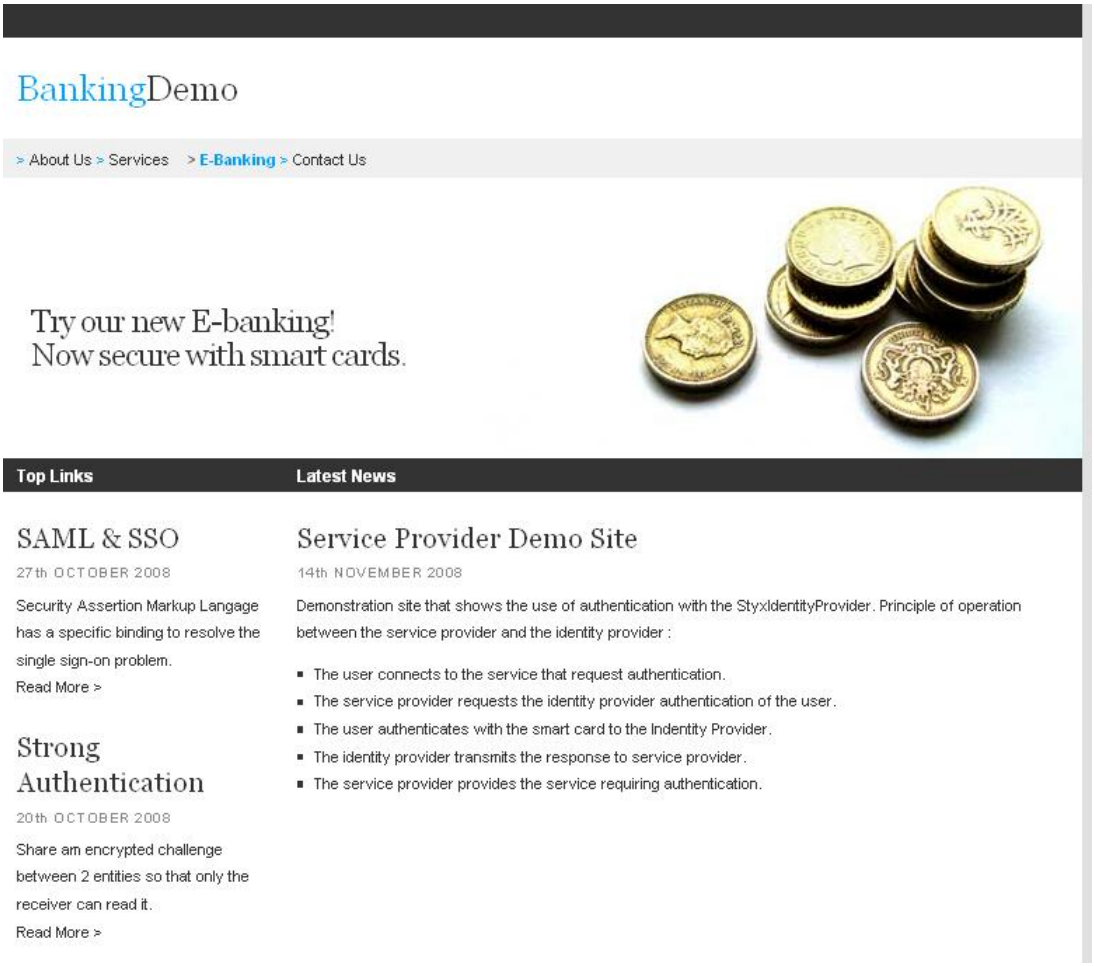
BankingDemo
> About Us > Services > E-Banking > Contact Us
Try our new E-banking!
Now secure with smart cards.
Top Links
Latest News
SAML & SSO
27th OCTOBER 2008
Security Assertion Markup Langage has a specific binding to resolve the single sign-on problem.
Read More >
Strong Authentication
20th OCTOBER 2008
Share am encrypted challenge between 2 entities so that only the receiver can read it.
Read More >
Service Provider Demo Site
14th NOVEMBER 2008
Demonstration site that shows the use of authentication with the StyxIdentityProvider. Principle of operation between the service provider and the identity provider :
The user connects to the service that request authentication.
The service provider requests the identity provider authentication of the user.
The user authenticates with the smart card to the Indentity Provider.
The identity provider transmits the response to service provider.
The service provider provides the service requiring authentication.

FIGURE 5.14 – Service Provider, implémentation, page d'accueil

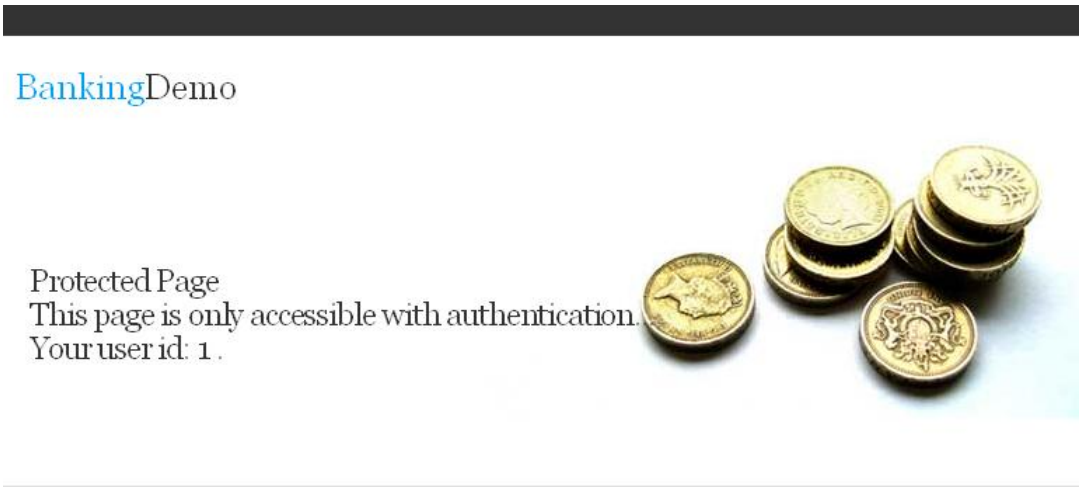

FIGURE 5.15 – Service Provider, implémentation, page protégée

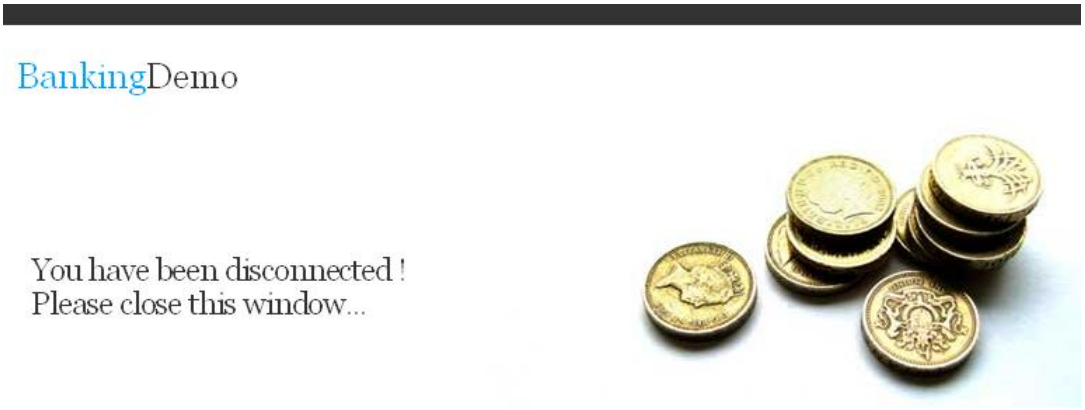

FIGURE 5.16 – Service Provider, implémentation, page de déconnexion

5.4.2 Chargement des clés dans keytool pour SSL

Pour utiliser SSL avec Tomcat du côté du fournisseur de service, il est nécessaire d'insérer la clé privée et le certificat publique dans le keytool de la machine où le serveur se trouve. Il est malheureusement impossible de faire ça manuellement avec l'outil keytool car il n'accepte que les importations de certificat publique. SSL a bien entendu aussi besoin de la clé privée. C'est pour cela que la page `install_keys.jsp` a été crée. Elle permet d'insérer la clé privée et le certificat du fournisseur de service dans son keytool. Ainsi SSL peut être utilisé avec les clés du fournisseur d'identité. Dans un premier temps, on récupère la clé et le certificat sur le serveur et ensuite on crée le nouveau keystore :

```
//Insert in key store
String password = (String) request.getParameter("pass");
String keystorename = System.getProperty("keystore");

if (keystorename == null)
keystorename = System.getProperty("user.home")
+ System.getProperty("file.separator")
+ ".keystore"; // especially this ;-)

KeyStore ks = KeyStore.getInstance("JKS", "SUN");
ks.load(null, password.toCharArray());

ks.store(new FileOutputStream(keystorename),
  password.toCharArray());
ks.load(new FileInputStream(keystorename),
  password.toCharArray());

java.security.cert.Certificate[] certs
  = new java.security.cert.Certificate[1];
certs[0] = (java.security.cert.Certificate)
  cf.generateCertificate(bis);

ks.setKeyEntry("StyxSP", (Key) pkCache.getPrivateKey(),
  password.toCharArray(), certs);

ks.store(new FileOutputStream(keystorename),
  password.toCharArray());
```

5.4.3 SAML

Envoi de la requête d'identification La requête d'authentification est envoyée avec la page JSP `request-auth.jsp`. Le toolkit SSO de Clareity Security est bien entendu utilisé, mais il a été modifié pour permettre la signature de la requête. Sa modification est abordée au sous-chapitre suivant. Dans un premier temps, on récupère les clés avec les classes `PrivateKeyCache` et `PublicKeyCache`. Ensuite on crée la requête avec tout les éléments nécessaires.

```
//To create a HTTP redirect
RedirectHandler redirectHandle = new RedirectHandler();

// The location of the recv auth request on the IdP
actionURL = application.getInitParameter("IdPRecvSAMLURL");

// Our unique name
issuerName = application.getInitParameter("SPName");

// Where receive response on this SP
assertionConsumerServiceURL =
  application.getInitParameter("SPRecvSAMLURL");
```

```
//Prepare & sending of the auth request
redirectHandle.setActionURL(actionURL);
redirectHandle.setAssertionConsumerServiceURL(
  assertionConsumerServiceURL);
redirectHandle.setIssuerName(issuerName);
redirectHandle.setProviderName(
  application.getInitParameter("IdPName"));
redirectHandle.setBindindUriFormat(
  RedirectHandler.POST_BINDING);
redirectHandle.setPrivateKeyCache(pkCache);
redirectHandle.setPublicKeyCache(pubCache);
```

Cette requête est ensuite envoyée par un HTTP Redirect vers le fournisseur d'identité en utilisant le "actionURL" et en appelant la méthode suivante où "response" est le `HttpServletResponse` :

```
redirectHandle.sendSAMLRedirect(response);
```

Réception de la réponse d'identification La réception des réponses aux requêtes d'authentification est prise en charge par la page `recv-saml.jsp`. Dans un premier, le fournisseur de service télécharge le fichier de metadata pour obtenir le certificat du fournisseur d'identité. Il sert à contrôler la requête de certification. Le téléchargement et le stockage des méta-données sont opérés grâce à la classe `metadataCache.java` du toolikt SSO de Clareity Security. Ensuite on reçoit la réponse et contrôle sa signature :

```
// Create our object to receive the response from the IdP
RecvResponse recvResponse = new RecvResponse();
// Attach our signature validator object to it
recvResponse.setSignatureValidator(metaDataCache.getSignatureValidator());
```

Finalement on vérifie la validité du message SAML en récupérant les paramètres transférés :

```
try {
  recvResponse.processRequest(request);
  loginId = recvResponse.getLoginId();
  relayState = recvResponse.getRelayState();
  messageXML = recvResponse.getResponseXML();
  session.setAttribute("userid", loginId);
  messageXML = messageXML.replace("<", "&lt;");
  messageXML = messageXML.replace(">", "&gt; ");
} catch (org.opensaml.xml.validation.ValidationException ve) {
   // if we get here, the assertion is not valid
}
```

5.4.4 Modification du toolkit SAML pour la signature des requêtes d'authentification

Le toolkit SSO de Clareity Security n'est vraiment pas complet. Il est en constante évolution et donne plus une manière d'utiliser OpenSAML pour faire de la SSO. Lors de son utilisation, une fonction importante manquait : il était impossible de signer les requêtes d'authentification envoyées par les Service Providers. Il était donc impossible pour le fournisseur d'identité de savoir avec certitude quel fournisseur de service le contactait et surtout,

si le message n'avait pas été intercepté et modifié par un tiers. Voici donc les modifications apportées au toolkit :

- `net.clareitysecurity.websso.sp.AbstractHttpHandler` :
  - Ajout de la clé privée et publique

```
private PrivateKeyCache privateKeyCache;
private PublicKeyCache publicKeyCache;

public PrivateKeyCache getPrivateKeyCache() {
return privateKeyCache;
}
public void setPrivateKeyCache(
  PrivateKeyCache privateKeyCache) {
this.privateKeyCache = privateKeyCache;
}
public PublicKeyCache getPublicKeyCache() {
return publicKeyCache;
}
public void setPublicKeyCache(
  PublicKeyCache publicKeyCache) {
this.publicKeyCache = publicKeyCache;
}
```

  - Création des objets nécessaires à la signature de la requête. Cette partie est reprise de la signature de la réponse d'authentification faite par le fournisseur d'identité. Ces objets sont créés dans cette méthode `buildAuthnRequest()`, car c'est là que la requête est préparée.

```
org.opensaml.xml.security.x509.BasicX509Credential credential = null;
org.opensaml.xml.signature.impl.KeyInfoImpl keyInfo = null;

// Set up the signing credentials if we have been given them.
if (privateKeyCache != null) {
try {
  org.opensaml.xml.signature.impl.SignatureBuilder signatureBuilder
   = new org.opensaml.xml.signature.impl.SignatureBuilder();
signature = signatureBuilder.buildObject();
credential = new org.opensaml.xml.security.x509.BasicX509Credential();
  credential.setPrivateKey(privateKeyCache.getPrivateKey());
if (publicKeyCache != null) {
  credential.setPublicKey(publicKeyCache.getPublicKey());
  KeyInfoBuilder keyInfoBuilder = new KeyInfoBuilder();
  keyInfo = (KeyInfoImpl) keyInfoBuilder.buildObject();
  KeyInfoHelper.addCertificate(keyInfo,
    publicKeyCache.getX509Certificate());
  signature.setKeyInfo(keyInfo);
  if (log.isDebugEnabled())
    log.debug("SAMLResponse.java - KeyInfo added to signature.");
}
signature.setSigningCredential(credential);
signature.setSignatureAlgorithm(
  SignatureConstants.ALGO_ID_SIGNATURE_RSA_SHA1 );
signature.setCanonicalizationAlgorithm(
```

```
    SignatureConstants.ALGO_ID_C14N_EXCL_OMIT_COMMENTS );
  } catch (Exception e) {
    e.printStackTrace();
  }
  }
```

  - Ajout de la signature à la requête. Cette opération se déroule toujours dans la même méthode.
    `auth.setSignature(signature);`
- `net.clareitysecurity.websso.sp.RedirectHandler` : Dans la méthode `sendSAMLRedirect()` il faut réellement donner l'ordre de signature après avoir sérialisé l'objet pour le transfert par HTTP Redirect. Cette création de signature s'obtient par l'instruction suivante : `org.opensaml.xml.signature.Signer.signObject(signature);`

5.5 Sécurisation des fournisseurs d'identité et de service

5.5.1 Tomcat et SSL

Pour sécuriser le serveur Tomcat, il faut au préalablement installer une clé privée et un certificat dans un keystore. On peut générer automatiquement un keystore avec la commande keytool mais attention la taille maximale des clés est de 1024bit. Il est aussi impossible d'importer une clé privée dans un keystore avec keytool. C'est pour ces raisons que l'on emploie directement le Service Provider `install_keys.jsp` ou l'Identity Provider (interface d'administration, Provider Settings administration) pour créer les keystores. Par contre il faut configurer les serveurs Tomcat. Pour se faire, il faut éditer le fichier `server.xml` et rajouter le code suivant :

```
<Connector
port="8443"
protocol="HTTP/1.1"
maxThreads="150"
SSLEnabled="true"
clientAuth="false"
keystorePass="styxsp"
keystoreFile="C:\Documents and Settings\olivid05\.keystore"
scheme="https"
secure="true"
sslProtocol="TLS"
/>
```

Dans ce code il faut prêter attention aux attributs qui dépendent de l'installation du keystore :
- `keystorePass` Le mot de passe du keystore.
- `keystoreFile` Le chemin où se trouve le keystore

5.5.2 Politique de sécurité pour l'accès au pages

Par défaut on peut accéder au pages présentes sur le serveur soit en HTTP soit en HTTPS. Il est intéressant de bloquer l'accès à certaines pages avec le mode HTTPS. Pour se faire il faut modifier le fichier `web.xml` présent dans le dossier `WEB-INF`.

Service Provider Voici les contraintes de sécurité minimum conseillées pour le Service Provider :

```
<security-constraint>
<web-resource-collection>
  <web-resource-name>request-auth.jsp</web-resource-name>
  <url-pattern>/request-auth.jsp</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
  <web-resource-name>recv-saml.jsp</web-resource-name>
```

```
  <url-pattern>/recv-saml.jsp</url-pattern>
  </web-resource-collection>
<user-data-constraint>
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
  <web-resource-name>protected.jsp</web-resource-name>
  <url-pattern>/protected.jsp</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Il s'agit de protéger les pages sensibles s'occupant de l'authentification et les pages qu'elle protège.

Identity Provider Voici les contraintes de sécurité minimum conseillées pour l'Identity Provider :

```
<security-constraint>
  <web-resource-collection>
  <web-resource-name>SecureApp</web-resource-name>
  <url-pattern>*.jsp</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Sur l'Identity Provider toutes les pages sont protégées car leur contenu doit rester confidentiel.

5.5.3 Protection de l'accès au page de gestion

L'accès aux pages de gestion est protégé depuis le code JSP, car aucune configuration n'est possible dans le fichier `web.xml` ou `server.xml`. Le code de protection suivant a été rajouté au début de chaque page :

```
//Restricted access to administration area
if(!(request.getRemoteAddr().equals("127.0.0.1")||
   request.getRemoteAddr().equals(
     getServletContext().getInitParameter("admin_ip")))){
  out.print("Unauthorized access");
  return;
}
```

Il est possible de préciser une autre adresse IP, qui est configuré dans le `web.xml`, pour l'administration :

```
<context-param>
  <param-name>admin_ip</param-name>
  <param-value>160.98.21.203</param-value>
</context-param>
```

5.6 Librairies

Librairies "endorsed" sont les librairies qui doivent être mises dans le répertoire `lib/endorsed` de la machine virtuelle utilisée par les serveurs.

- resolver-2.9.1.jar
- serializer-2.9.1.jar
- xalan-2.7.1.jar
- xercesImpl-2.9.1.jar
- xml-apis-2.9.1.jar

Librairies Voici les autres librairies utilisées par les serveurs :

- bcprov-jdk15-141.jar
- commons-codec-1.3.jar
- commons-collections-3.1.jar
- commons-lang-2.1.jar
- httpclient-3.0.1-commons-codecs-1.3.jar
- httpclient-3.0.1.jar
- jargs-1.0.jar
- javolution.jar
- jcl104-over-slf4j-1.4.3.jar
- joda-time-1.5.1.jar
- log4j-1.2.14.jar
- log4j-over-slf4j-1.4.3.jar
- mysql-connector-java-5.0.8-bin.jar
- not-yet-commons-ssl-0.3.9.jar
- opensaml.jar
- openws.jar
- slf4j-api-1.4.3.jar
- slf4j-log4j12-1.4.3.jar
- velocity-1.5.jar
- xmlsec-1.4.1-commons-logging.jar
- xmlsec-1.4.1.jar
- xmltooling.jar

La librairie du toolkit SSO de Clareity Security est directement intégrée au code de l'application pour faciliter sa compréhension. De plus la version jar ne comprend pas les dernières évolutions.

Conclusion

L'implémentation s'est déroulée avec succès malgré quelques fonctionnalités qui ont du être abandonnées faute de temps : par exemple, le stockage et traitement des certificats dans la carte. Il faut relever que le toolkit SSO de Clareity Security était très basique. Il a du être modifié pour proposer de nouvelles fonctionnalités. Il n'est pas remis en cause car il simplifie grandement l'utilisation de la librairie OpenSAML qui doit être utilisée pour travailler correctement (en suivant le standard) avec SAML sur la plateforme Java.

6 Validation et tests de sécurité

Introduction

Les tests de l'application, qu'ils soient fait en cours de codage ou sa la fin, permettent de vérifier son fonctionnement (ses fonctionnalités). Tous les tests se sont déroulés sur des PC avec les spécifications suivantes :

- Intel Core 2 DUO 6400 2.13GHz
- 2 Go de mémoire RAM
- Microsoft Windows XP SP3
- Firefox 3
- Pilote SCR3310 installé
- Java JDK et JRE 1.5 Update 8
- MySQL Community Server 5.0.67 (si serveur Identiry Provider)
- Apache Tomcat 6.0.18 (si serveur)

Il y a plusieurs types de test qui ont été effectués : les premiers sont les tests durant le développement de l'application pour chaque fonctionnalité ou partie de l'application, les seconds sont les tests fonctionnels sur un déploiement d'une première version de l'application. Ces tests sont chargés de tester toutes les fonctionnalités de l'application. Pour les finaliser, des analyses de sécurité sur l'infrastructure, sur certaines parties du prototype et sur le médium (données transférées) ont été effectuées.

6.1 Validation durant le développement

Les tests durant le développement sur un logiciel utilisant beaucoup de composant ne sont pas facile à mettre en place. Chaque composant n'a pas été testé séparément car ils interagissent tous entre-eux. Il aurait été possible d'émuler les composants nécessaires, mais cela revenait à les créer aussi.

Applet NXP JCOP

Une première version sans communication avec en applet WEB, mais avec une application standalone a été créée :
Application dans la carte :

```
public class test extends Applet implements SampleInterface {
private Dispatcher  dispatcher;
private KeyPair  kp;
private RSAPrivateCrtKey  kpriv;
private RSAPublicKey  kpub;
private byte[] challenge;
private byte[] response;
private test() {
CardRemoteObject.export(this);
dispatcher = new Dispatcher((short) 1);
dispatcher.addService(new RMIService(this),
Dispatcher.PROCESS_COMMAND);
}

public static void install(byte[] bArray,
short bOffset, byte bLength) {
new test().register(bArray, (short)
(bOffset + 1), bArray[bOffset]);
}

public void process(APDU apdu) {
dispatcher.process(apdu);
}

public void genKey() throws RemoteException {
kp= new KeyPair(KeyPair.ALG_RSA_CRT,(short)2048);
kp.genKeyPair();
kpriv=(RSAPrivateCrtKey)kp.getPrivate();
kpub=(RSAPublicKey)kp.getPublic();
}

public byte[] getPublicKeyExponent() throws RemoteException {
byte[] buffer = new byte[10];
short len=kpub.getExponent(buffer, (short)0);
byte[] res= new byte[len];
for(short i=0;i<res.length;i++)res[i]=buffer[i];
return res;
}
```

```
public byte[] getPublicKeyModulusFHalf() throws RemoteException {
byte[] buffer = new byte[256];
kpub.getModulus(buffer, (short)0);
byte[] ret = new byte[128];
for(short i=0; i< 128;i++)ret[i]=buffer[i];
return ret;
}

public byte[] getPublicKeyModulusLHalf() throws RemoteException {
byte[] buffer = new byte[256];
kpub.getModulus(buffer, (short)0);
byte[] ret = new byte[128];
for(short i=0; i< 128;i++)ret[i]=buffer[i+128];
return ret;
}


public byte[] process() throws RemoteException {
Cipher cipher = Cipher.getInstance(Cipher.ALG_RSA_PKCS1, true);
cipher.init(kpriv, Cipher.MODE_DECRYPT);
byte[]t = new byte[256];
short l=cipher.doFinal(challenge, (short)0,
(short)challenge.length, t, (short)0);
byte[] res = new byte[l];
for(short i=0;i<l;i++){
res[i]=t[i];
}
return res;
}

public void sendChallengeFPart(byte[] tmp) throws RemoteException {
challenge=new byte[256];
for(short i=0;i<128;i++){
challenge[i]=tmp[i];
}
}

public void sendChallengeLPart(byte[] tmp) throws RemoteException {
for(short i=0;i<128;i++){
challenge[i+128]=tmp[i];
}
}
}
```

Voici ensuite l'application de test standalone :

```
public class Test {
public static void main(String[] args) {
Security.addProvider(
    new org.bouncycastle.jce.provider.BouncyCastleProvider());
```

```
byte                            //Applet AID
//i 8899889981
AID[] = {(byte) 0x88,(byte) 0x99,(byte) 0x88,(byte) 0x99,(byte) 0x81};
JCTerminal term;

term = JCTerminal.getInstance("pcsc:4", null);
term.open();
JCard card = new JCard(term,null,5000);
JCApplet jca = new JCApplet(card,AID);
jca.select();
rsatest.SampleInterface obj =
  (rsatest.SampleInterface)RMIObjectFactory.getInitialReference(jca);

byte[] tocrypt = new byte[128];
for(int i=0; i<tocrypt.length;i++)tocrypt[i]=(byte)i;
System.out.println("CLEAR:"+Arrays.toString(tocrypt));
try{
obj.genKey();
RSAPublicKey pk = getPublicKey(obj);
Cipher cipher = Cipher.getInstance("RSA/None/PKCS1Padding");
cipher.init(Cipher.ENCRYPT_MODE, pk);
byte[]crypted = cipher.doFinal(tocrypt);
System.out.println("CRYPT:"+Arrays.toString(crypted));
byte[]ch_f = new byte[128];for(int i=0;i<128;i++)ch_f[i]=crypted[i];
byte[]ch_l = new byte[128];for(int i=0;i<128;i++)ch_l[i]=crypted[i+128];
obj.sendChallengeFPart(ch_f);
obj.sendChallengeLPart(ch_l);
byte[] decrypt=obj.process();
System.out.println("DECRY:"+Arrays.toString(decrypt));
}catch(Exception e){
e.printStackTrace();
}

}
protected static RSAPublicKey getPublicKey(rsatest.SampleInterface obj)
                                                    throws Exception {
byte[] exp=obj.getPublicKeyExponent();
byte[] fh=obj.getPublicKeyModulusFHalf();
byte[] lh=obj.getPublicKeyModulusLHalf();
byte[] mod= new byte[256];
for(int i=0; i<128;i++)mod[i]=fh[i];
for(int i=0; i<128;i++)mod[i+128]=lh[i];
BigInteger modulus = new BigInteger(1,mod);
BigInteger exponent = new BigInteger(1, exp);
RSAPublicKeySpec rpks = new RSAPublicKeySpec(modulus, exponent);
KeyFactory kf = KeyFactory.getInstance("RSA");
return (RSAPublicKey)kf.generatePublic(rpks);
}

}
```

Cette première version a permis de tester le cryptage et décryptage on et off card.

Applet WEB Administration

Voici la liste des tests :
- Test de la présence d'un code PIN
- Test de la longueur du code PIN et des chiffres uniquement
- Test de connexion à chaque opération de la carte avec le débugger `StyxJCConnector`
- Test de connexion à chaque opération sur le serveur d'identité avec le débugger `StyxIdPConnector`
- Installation du CAP File test en externe puis en interne (implémenté dans l'applet d'administration)
- Installation du driver et du CAP File sur le poste testé avec le débugger

Applet WEB Authentification

Voici la liste des tests :
- Test de la présence d'un code PIN
- Test de la longueur du code PIN et des chiffres uniquement
- Test de la modification du code PIN
- Test du transport des challenges/reponse
- Test de connexion à chaque opération de la carte avec le débugger `StyxJCConnector`
- Test de connexion à chaque opération sur le serveur d'identité avec le débugger `StyxIdPConnector`
- Test de durée : 500 authentification à la suite sans aucune erreur, environ 10 minutes.
- Test de l'envoi du message de logout
- Test de la détection du logout

Identity Provider

Voici la liste des tests :
- Test de la classe `SecurityFunctionsProvider`
- Test de la classe `RegisteredSPManager`
- Test de la classe `IdentityProviderManager`
- Test du servlet `download.java` : téléchargement de tous les fichiers depuis Firefox ou Internet Explorer et contrôle de corruption des données.
- Test du servlet `auth.java` : Essai d'obtention de challenge, de renvoi de réponse, de réception de logout et d'ouverture d'une connexion au Service Provider ou non depuis un navigateur Internet ou depuis l'applet d'authentification durant le codage.
- Test du servlet `administration.java` : Envoi de requête de certificat, obtention de certificat du IdP, obtention des clés du CardManager depuis un navigateur Internet ou depuis l'applet d'administration durant le codage.
- Test du contrôle des exceptions sur chaque page JSP.
- Test de tous les liens et bouton présents.
- Test du login de l'espace d'administration (ressemble au test fonctionnel)
- Test de la protection des pages d'administration
- Test d'ajout suppression modification d'utilisateur (ressemble au test fonctionnel)
- Test de toutes les fonctions d'administration (ressemble au test fonctionnel)
- Test de toutes les fonctions relatives au Service Provider
- Test de communication SAML et traitement avec le débugger

– Test des fonctions de login/logout d'utilisateur et pérennité de l'authentification avec la page `login.jsp` et le débugger sur le serveur.

Service Provider

Voici la liste des tests :
– Test de traitement SAML avec le débugger sur le serveur.
– Test de réception de fausse authentification SAML
– Test de la pérennité de l'authentification avec Java script (logout du Service Provider)
– Test de la protection des pages sécurisées.

6.2 Validation fonctionnelle

La validation consiste d'abord à déployer l'infrastructure de test et à tester toutes les fonctionnalités en utilisant l'infrastructure au complet pour les tests de concurrence par exemple. L'infrastructure de test est constituée de 3 machines ayant les caractéristiques techniques cité précédemment : 2 fournisseurs de service et un fournisseur d'identité. Un des fournisseurs de service fait aussi office de client. La validation fonctionnelle est divisée en 3 parties : la validation du fournisseur d'identité, du Service Provider, et de l'infrastructure globale.

6.2.1 Test des fonctions de l'Identity Provider

Installation

Situation initiale : aucun programme n'est installé sur l'ordinateur.

Scénario :

1. Adresse IP : 192.168.0.1
   Nom du PC : StyxIdP
2. Installation de MySQL 5.0 avec utilisateur local "root" et mot de passe "dipl08" dans `d:\MySQL\`
3. Installation de Java Developement Kit dans `d:\jdk15`
4. Installation de Tomcat dans `c:\Program File\Apache Software Foundation\Tomcat 6.0` , utilisateur "admin" et mot de passe "dipl08".
5. Copie des librairies "endorsed" dans `\Tomcat 6.0\endorsed`.
6. Installation de `styxdb.sql` avec la commande suivante :
   `mysql -u root -p <styxdb.sql`
7. Déploiement de l'application `StyxWebSecurity_IdP.war` à l'aide du Tomcat Manager
8. Dans le `web.xml` passez de confidential à none
   `dataBaseUser` = root
   `dataBasePassword` = dipl08
   `IdPName` = StyxIdP
   `IdPCheckAuth` = https://StyxIdP:8443/StyxWebSecurity_IdP/check_auth.jsp
   `IdPRecvSAMLURL` = https://StyxIdP:8443/StyxWebSecurity_IdP/recv_saml_auth_request.jsp
   `admin_ip` = 192.168.1.1
9. Connexion avec IE7 sur http://localhost:8080/StyxWebSecurity_IdP/admin.jsp
10. Dans l'espace d'administration le création du keytool,le keystore se trouve dans le dossier utilisateur.
11. Modification de server.xml pour SSL
12. Dans le `web.xml` passez de none à confidential
13. Signature des jars (dans Applets) avec le keystore
14. Accès à l'espace d'administration en SSL
15. Contrôle des signature des applets

Résultat : L'installation s'est déroulée avec succès. Il faut modifier le nom du certificat par celui du nom de la machine dans le web.xml et non pas par le nom fixe actuel lors de la génération, cela nécessite une nouvelle version de la génération de certificat.

Changement du mot de passe administrateur

Situation initiale : L'Identity Provider est installé par défaut.

Scénario :

1. Connexion à l'espace d'administration de l'Identity Provider
2. Dans l'espace d'administration, dans "Provider"
3. Modification du mot de passe "dipl08"
4. Déconnexion
5. Re-connexion avec le nouveau mot de passe

Résultat : Le test s'est déroulé avec succès, il est possible de changer facilement de mot de passe.

IdP Modification de chaque une des valeurs d'un utilisateur et visualisation

Situation initiale : L'Identity Provider est installé par défaut.

Scénario :

1. Connexion à l'espace d'administration du serveur d'identification
2. Modification de l'utilisateur John Smith
3. Changement de son prénom et validation
4. Visualisation du changement, ok
5. Modification de l'utilisateur John Smith
6. Changement de son nom et validation
7. Visualisation du changement, ok
8. Modification de l'utilisateur John Smith
9. Changement de son adresse et validation
10. Visualisation du changement, ok
11. Modification de l'utilisateur John Smith
12. Changement de sa ville et validation
13. Visualisation du changement, ok
14. Modification de l'utilisateur John Smith
15. Changement de son numéro postal et validation
16. Visualisation du changement, ok
17. Modification de l'utilisateur John Smith
18. Changement de son pays et validation
19. Visualisation du changement, ok
20. Modification de l'utilisateur John Smith
21. Changement de son téléphone et validation
22. Visualisation du changement, ok
23. Modification de l'utilisateur John Smith
24. Changement de son email et validation
25. Visualisation du changement, ok

Résultat : Le test s'est déroulé avec succès.

IdP Logout de l'espace d'administration

Situation initiale : L'Identity Provider est installé par défaut.

Scénario :

1. Connexion dans l'espace d'administration
2. Déconnexion de l'espace d'administration
3. Tentative infructueuse de ré-entrer dans l'espace sans se re-connecter.

Résultat : Le test s'est déroulé avec succès.

IdP Logout d'un utilisateur

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario :

1. Sur la page de login de l'Identity Provider
2. Insertion de la carte
3. Insertion du PIN
4. Continuer l'identification
5. L'utilisateur est authentifié
6. Retrait de la carte
7. L'utilisateur est directement déconnecté

Résultat : Le test s'est déroulé avec succès.

SSL forcé sur les pages

Situation initiale : L'Identity Provider est installé par défaut.

Scénario :

1. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/admin.jsp redirigé en SSL
2. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/login.jsp redirigé en SSL
3. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/check_auth.jsp redirigé en SSL
4. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/login_control.jsp redirigé en SSL
5. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/recv_saml_auth_request.jsp redirigé en SSL
6. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/saml_auth.jsp redirigé en SSL
7. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/provider_admin.jsp redirigé en SSL
8. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/sp_add.jsp redirigé en SSL

9. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/sp_del.jsp redirigé en SSL
10. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/sp_view.jsp redirigé en SSL
11. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_add.jsp redirigé en SSL
12. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_del.jsp redirigé en SSL
13. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_gsc.jsp redirigé en SSL
14. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_man.jsp redirigé en SSL
15. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_mod.jsp redirigé en SSL
16. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_view.jsp redirigé en SSL

Résultat : Toutes les pages attendues ont été redirigées avec succès en SSL.

Espace de management et IP qui y accède

Situation initiale : L'Identity Provider est installé par défaut.

Scénario : Connexion sur les pages suivantes depuis le poste du client (son ip n'est pas égal à l'`admin_ip` du `web.xml`, une message d'erreur devrait apparaître.

1. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/admin.jsp
2. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/provider_admin.jsp
3. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/sp_add.jsp
4. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/sp_del.jsp
5. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/sp_view.jsp
6. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_add.jsp
7. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_del.jsp
8. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_gsc.jsp
9. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_man.jsp
10. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_mod.jsp
11. Connexion sur http://styxidp:8080/StyxWebSecurity_IdP/Management/user_view.jsp

Résultat : Impossible d'accéder à toutes ces pages. Le message d'erreur est "Unauthorised access". Le test est donc passé avec succès.

Mot de passe administrateur incorrect

Situation initiale : L'Identity Provider est installé par défaut.

Scénario :

1. Connexion sur l'espace d'administration de l'Identity Provider avec un mot de passe vide
2. Erreur pas possible
3. Connexion sur l'espace d'administration de l'Identity Provider avec un mot de passe faux
4. Erreur pas possible
5. Connexion sur l'espace d'administration de l'Identity Provider avec un mot de passe faux très grand
6. Erreur max 30 de longueur
7. Connexion sur l'espace d'administration de l'Identity Provider avec utilisateur vide
8. Erreur utilisateur obligatoire
9. Connexion sur l'espace d'administration de l'Identity Provider avec faux utilisateur et mot de passe juste
10. Erreur pas possible

Résultat : Le test a été passé avec succès.

Modification des Card Manager Keys, recréation des cartes

Situation initiale : L'Identity Provider est installé par défaut.

Scénario :

1. Connexion sur l'espace d'administration de l'Identity Provider
2. Dans "Provider"
3. Dans CardManagerKey 16x1 devient 16xA
4. On re-génère une carte pour un utilisateur
5. Le programme de demande les clés du CM donc 16x1, 16x2 et 16x3
6. La carte est générée.
7. On modifie dans "Provider" et remet 16x1
8. On re-génère la même carte
9. Le programme demande les clés du CM donc 16xA, 16x2 et 16x3

Résultat : Le test est passé avec succès. Si le programme n'a pas les clés du CardManager, il les demande (à part si les clés sont les clés JCOP par défaut).

Formulaire d'insertion d'utilisateur

Situation initiale : L'Identity Provider est installé par défaut.

Scénario : Zones de texte requises :

1. First Name
2. Last Name
3. Email

Zones de texte avec longueurs maximale :

1. First Name
2. Last Name
3. Address
4. Zip
5. City
6. Country
7. Phone
8. Email

Zones de texte typées :

1. Adresse
2. Zip
3. Phone
4. Email

Résultat : La longueur maximale du champs adresse n'est pas contrôlée.

Formulaire de modification d'utilisateur

Situation initiale : L'Identity Provider est installé par défaut.

Scénario : Zones de texte requises :

1. First Name
2. Last Name
3. Email

Zones de texte avec longueurs maximale :

1. First Name
2. Last Name
3. Address
4. Zip
5. City
6. Country
7. Phone
8. Email

Zones de texte typées :

1. Adresse
2. Zip
3. Phone
4. Email

Résultat : La longueur maximale du champs adresse n'est pas contrôlée.

Formulaire d'ajout de Service Provider

Situation initiale : L'Identity Provider est installé par défaut.

Scénario :

1. Dans l'espace d'administration sous "Registered Service Provider", puis "Add"
2. Le champs est bien obligatoire
3. Le champs ne contient pas d'espace
4. Le champs a une longueur max de 50

Résultat : Le test s'est déroulé avec succès.

Génération d'une nouvelle paire de clés IdP et certificat

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario :

1. Dans l'espace d'administration de l'Identity Provider sous "Provider"
2. "Generate new keys"
3. ensuite "Start generation of new certificate"
4. Ecrasement du keytool avec les nouvelles clés
5. Redémarrage du serveur
6. Connexion SSL, revalidation du certificat
7. Essai d'utilisation d'une ancienne carte, impossible
8. Création d'une nouvelle carte
9. Connexion avec la nouvelle carte réussie.

Résultat : Le test s'est déroulé avec succès.

6.2.2 Test des fonctions du Service Provider

Installation

Situation intiale : aucun programme n'est installé sur l'ordinateur.

Scénario :

1. Adresse IP : 192.168.1.2
   Nom du PC : StyxSPBankDemo
2. Installation de Java Developement Kit dans `d:\jdk15`
3. Installation de Tomcat dans `c:\Program File\Apache Software Foundation\Tomcat 6.0` , utilisateur "admin" et mot de passe "dipl08".
4. Copie des librairies "endorsed" dans `\Tomcat 6.0\endorsed`.

5. Déploiement de l'application `StyxWebSecuritySP.war` à l'aide du Tomcat Manager
6. Inscription du Service Provider dans l'Identity Provider (obtention du certificat, private key et public key) avec le nom de la machine.
7. Modification du `web.xml`
   `IdPMetadataFileURL` http://StyxIdP:8080/StyxWebSecurity_IdP/Management/download?type=meta
   `IdPLoginControlURL` https://StyxIdP:8443/StyxWebSecurity_IdP/login_control.jsp
   `IdPRecvSAMLURL` https://StyxIdP:8443/StyxWebSecurity_IdP/recv_saml_auth_request.jsp
   `IdPName` StyxWebSecurityIdentityProvider
   `SPRecvSAMLURL` https://StyxSPBankDemo:8443/StyxServiceProvider/recv-saml.jsp
   `SPName` StyxSPBankDemo
8. Copie des clés dans `c:\`, création d'un keystore avec `install_keys.jsp`
9. Configuration de SSL sur Tomcat dans `server.xml`
10. Accès au Service Provider avec ou sans SSL

Résultat : L'installation s'est bien passée mais une erreur a été découverte dans le certificat du Service Provider qui ne retrouve pas sa chaîne de certification car le nom de l'issuer n'est pas "StyxWebSecurityIdentityProvier" mais "StyxWebSecurity Identity Provier". Il faut aussi corriger cette erreur.

SP Logout d'un utilisateur

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario :

1. Connexion sur le Service Provider
2. Demande de connexion dans l'espace "e-banking"
3. Transmission de la requête d'authentification à l'Identity Provider
4. Insertion de la carte
5. Insertion du PIN
6. Continuer l'identification
7. Affichage de l'espace protégé
8. Attente du 3 minutes de navigation
9. Retrait de la carte
10. Apparition du message de déconnexion, maximum 10 seconde après le retrait de la carte
11. Affichage de la page de déconnexion du fournisseur de service
12. Impossible d'accéder à la page protégée d'avant

Résultat : Le test s'est déroulé avec succès.

SP SSL forcé sur les pages

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario : Le test consiste à ce connecter sur les pages suivantes et voir si elles sont redirigées dans le mode SSL :

1. http://StyxSPBankDemo/StyxServiceProvider/request-auth.jsp
2. http://StyxSPBankDemo/StyxServiceProvider/recv-saml.jsp
3. http://StyxSPBankDemo/StyxServiceProvider/protected.jsp

Résultat : Toutes les pages ont été redirigées. Le test est un succès.

Fausses clés données au Service Provider

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario :

1. Remplacement des clés ne correspondant pas au Service Provider
2. Essai d'authentification
3. Message d'erreur "Authentification request, signature not valid"

Résultat : Le test s'est déroulé avec succès.

Suppression d'un Service Provider et authentification

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario :

1. Dans l'interface d'administration, suppression du deuxième Service Provider.
2. Essai de connexion sur ce deuxième Service Provider
3. Message d'erreur "Service Provider not found or certificate not valid"

Résultat : Le test s'est déroulé avec succès.

6.2.3 Test des fonctions de l'applet On-card et applet d'authentification ou d'administration

Modification PIN

Situation initiale : L'Identity Provider est installé par défaut. Un utilisateur et sa carte est créé.

Scénario :

1. Connexion sur l'Identity Provider sur la page de login
2. Insertion de la carte
3. Insertion du PIN d'installation 1234
4. Clique sur "Modify PIN"
5. Insertion du PIN 9876

6. Redémarrage du navigateur et connexion sur la page login
7. Insertion du PIN 1234
8. Réception d'un message d'erreur, ce qui est normal
9. Insertion du PIN 9876
10. L'utilisateur est logué.

Résultat : Le test s'est déroulé avec succès.

Blocage de la carte après 3 PIN faux

Situation initiale : L'Identity Provider est installé par défaut. Un utilisateur et sa carte est créé.

Scénario :

1. Connexion sur l'Identity Provider sur la page de login
2. Insertion de la carte
3. Insertion du PIN faux 9999
4. Insertion du PIN faux 9999
5. Insertion du PIN faux 9999
6. Insertion du PIN vrai 1234
7. Le PIN vrai n'est pas accepté.

Résultat : Le test s'est déroulé avec succès.

Authentification sans carte

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario :

1. Connexion sur l'Identity Provider sur la page de login
2. Insertion du PIN
3. L'authentification est refusée.

Résultat : Le test s'est déroulé avec succès, mais il faudrait afficher un message qui demande l'insertion de la carte.

Génération d'un carte

Situation initiale : L'Identity Provider est installé par défaut.

Scénario :

1. Connexion sur l'espace d'administration
2. Création d'une carte pour un utilisateur
3. Connexion avec la carte sur la page de login

Résultat : Le test s'est déroulé avec succès.

6.2.4 Test du système dans sa globalité

Création de 3 utilisateurs (cartes comprises) et authentification

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario :

1. Création de 3 utilisateurs avec cartes : John Smith / Jane Smith / Marcel Durant.
2. Connexion de Jane Smith, Autoriser les popups depuis l'Identity Provider
3. Connexion de Jane Smith, accès au service protégé, déconnexion correcte.
4. Redémarrage du navigateur
5. Connexion de John Smith, accès au service protégé, déconnexion correcte.
6. Redémarrage du navigateur
7. Connexion de Marcel Durant, accès au service protégé, déconnexion correcte.

Résultat : Le test s'est déroulé correctement. Il faudrait que lorsqu'on tape sur entré, l'action de login s'active automatiquement et que l'on aie pas le besoin de cliquer.

Création de 2 utilisateurs (cartes comprises) et authentification en même temps

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut. Ce test est limité à 2 utilisateur, car nous n'avons que 2 lecteurs de carte.

Scénario :

1. Jane Smith se connecte sur le Service Provider.
2. Marcel Durant se connecte sur le Service Provider.
3. Les 2 utilisateurs sont connectés correctement sur 2 PCs différents et sur le même Service Provider.
4. Retrait de la carte de Marcel Durant, il se déconnecte. Jane Smith est toujours connectée.
5. Retrait de la carte de Jane Smith, elle se déconnecte correctement.

Résultat : Le test s'est déroulé avec succès.

Suppression d'un utilisateur et authentification

Situation initiale : L'Identity Provider est installé par défaut. Un Service Provider est installé par défaut.

Scénario :

1. Connexion sur l'espace d'administration de l'Identity Provider
2. Suppression de Marcel Durant
3. Sur le poste de client, connexion au Service Provider
4. Insertion de la carte de Marcel Durant
5. Insertion du code PIN de Marcel Durant
6. Connexion à la carte ok et proposition de changement de PIN
7. Continuer l'authentification
8. Message d'erreur : "Incorrect identification, please contact our administrator"

Résultat : Le test s'est déroulé avec succès.

Création de 2 Service Providers avec 1 utilisateur sur les 2

Situation initiale : L'Identity Provider est installé par défaut. Un premier Service Provider est installé par défaut. Un second Service Provider est installé par défaut.

Scénario :

1. Connexion au premier Service Provider
2. Authentification auprès de l'Identity Provider
3. Affichage de la page sécurisée
4. Connexion dans un nouvelle fenêtre au 2 ième Service Provider
5. Affichage direct de la page sécurisé car le client est déjà authentifié au fournisseur d'identité

Résultat : Le test s'est déroulé avec succès.

Création de 2 Service Providers avec 2 utilisateurs sur les 2

Situation intiale : L'Identity Provider est installé par défaut. Un premier Service Provider est installé par défaut. Un second Service Provider est installé par défaut.

Scénario :

1. Utilisateur 1, Connexion au premier Service Provider
2. Utilisateur 1, Authentification auprès de l'Identity Provider
3. Utilisateur 1, Affichage de la page sécurisée
4. Utilisateur 1, Connexion dans un nouvelle fenêtre au 2 ième Service Provider
5. Utilisateur 1, Affichage direct de la page sécurisé car le client est déjà authentifié au fournisseur d'identité
6. Utilisateur 2, Connexion au premier Service Provider
7. Utilisateur 2, Authentification auprès de l'Identity Provider
8. Utilisateur 2, Affichage de la page sécurisée
9. Utilisateur 2, Connexion dans un nouvelle fenêtre au 2 ième Service Provider

10. Utilisateur 2, Affichage direct de la page sécurisé car le client est déjà authentifié au fournisseur d'identité
11. Utilisateur 1, déconnexion
12. Utilisateur 2, déconnexion

Résultat : Le test s'est déroulé avec succès.

Logout d'un utilisateur de 2 Service Providers

Situation initiale : L'Identity Provider est installé par défaut. Un premier Service Provider est installé par défaut. Un second Service Provider est installé par défaut.

Scénario :

1. Connexion avec un utilisateur à 2 Service Providers
2. Retrait de la carte.
3. Les Service Providers sont déconnectés dans les 10 secondes

Résultat : Le test s'est déroulé avec succès.

6.2.5 Résultats de la validation fonctionnelle

La plupart des tests se sont déroulés avec succès. Une grande partie des fonctions de l'application ne posent aucun problème. Mais il reste des petites modifications à effectuer pour augmenter la qualité du prototype :

- Génération des certificats avec le nom `IdPName` qui se trouve dans le `web.xml`, cela pour éviter d'utiliser les certificats pour signer les applets ou SSL alors qu'ils ne correspondent pas au nom d'hôte.
- Modification de l'applet d'authentification et d'administration : pour que lorsque'on tape sur entrer on continue automatiquement (lorsqu'on saisi le PIN).
- Modification de l'applet d'authentification et d'administration : pour afficher un message d'erreur lorsque la carte n'est pas présente.
- Modification des pages `user_add.jsp` et `user_mod.jsp` : pour que la longueur de l'adresse soit contrôlée correctement.

Ces dernières modifications ont été rajoutées dans la version 1.1 du prototype. Elles ont toutes été contrôlées.

6.3 Tests de sécurité du fournisseur d'identité avec la méthode OSSTMM

L'analyse de sécurité ne comporte que l'analyse du fournisseur d'identité. Un réseau de test à été mis en place, il ne comporte ni firewall ni IDS, ces éléments ne seront donc pas analysé. Les méthodes de Social Engineering ne sont pas testées car aucune infrastructure de personnel utilisant les système est en place (pas de personne en rapport avec le système). Les mesures de quarantaine et de détection de virus ne sont pas utiles dans le cas du test de l'Identity Provider, car on cherche en premier lieu à tester le prototype. De même aucun envoi ou réception d'email est géré par le serveur, ces tests sont donc inutiles.

6.3.1 Profil réseau

Plage IP de test 192.168.1.1-192.168.1.3

Informations de domaine et configuration aucune domaine tous dans le Workgroup STYXWEBSECURITY

DNS pas de serveur DNS, réseau de test non connecté à Internet

Liste des serveur

1. 192.168.1.1, StyxIdP, Windows XP SP3
2. 192.168.1.2, StyxSPBankDemo, Windows XP SP3
3. 192.168.1.3, vecteur d'attaque dans le sous-réseau.

6.3.2 StyxIdP, information sur le serveur

Adresse IP 192.168.1.1

Nom réseau StyxIdP

Liste des ports ouverts Avec le logiciel NMAP :

```
david@master-laptop:~$ sudo nmap -sS 192.168.1.1

Starting Nmap 4.53 ( http://insecure.org ) at 2008-11-04 15:19 CET
Interesting ports on 192.168.1.1:
Not shown: 1706 closed ports
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
3306/tcp open  mysql
8009/tcp open  ajp13
8080/tcp open  http-proxy
8081/tcp open  blackice-icecap
8443/tcp open  https-alt
MAC Address: 00:19:D1:25:2D:5C (Intel)

Nmap done: 1 IP address (1 host up) scanned in 15.078 seconds
```

Liste des entêtes Avec le logiciel NMAP :

```
david@master-laptop:~$ sudo nmap -sV 192.168.1.1

Starting Nmap 4.53 ( http://insecure.org ) at 2008-11-04 15:21 CET
Interesting ports on 192.168.1.1:
Not shown: 1706 closed ports
PORT     STATE SERVICE      VERSION
135/tcp  open  msrpc        Microsoft Windows RPC
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds Microsoft Windows XP microsoft-ds
3306/tcp open  mysql        MySQL (unauthorized)
8009/tcp open  ajp13?
8080/tcp open  http         Apache Tomcat/Coyote JSP engine 1.1
8081/tcp open  http         Network Associates ePolicy
          Orchestrator (Computername: STYXIDP Version: 3.6.0.574)
8443/tcp open  ssl/http     Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 00:19:D1:25:2D:5C (Intel)
Service Info: OS: Windows

Host script results:
|_ Discover OS Version over NetBIOS and SMB: Windows XP

Service detection performed. Please report
any incorrect results at http://insecure.org/nmap/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 56.442 seconds
```

fingerprint Avec l'analyse "nmap" précédente, on voit que Windows XP est le système d'exploitation de la machine. Avec Xprobe2, le résultat est plus flou mais on constate quand-même que le système d'exploitation est Windows :

```
bt ~ # xprobe2 192.168.1.1

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu,
ofir@sys-security.com, meder@o0o.nu

[+] Target is 192.168.1.1
[+] Loading modules.
[+] Following modules are loaded:
...
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
...
[+] Primary guess:
[+] Host 192.168.1.1 Running OS:
"Microsoft Windows 2003 Server Standard Edition" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.1.1 Running OS:
"Microsoft Windows 2003 Server Enterprise Edition" (Guess probability: 100%)
[+] Host 192.168.1.1 Running OS:
"Microsoft Windows XP SP2" (Guess probability: 100%)
```

```
[+] Host 192.168.1.1 Running OS:
"Microsoft Windows 2000 Workstation" (Guess probability: 100%)
...
[+] Host 192.168.1.1 Running OS:
"Microsoft Windows 2000 Server" (Guess probability: 100%)
...
[+] Execution completed.
```

Ensuite avec "httpprint" : pour identifier le serveur WEB, le résultat devient plus intéressant même si la version n'est pas correcte :

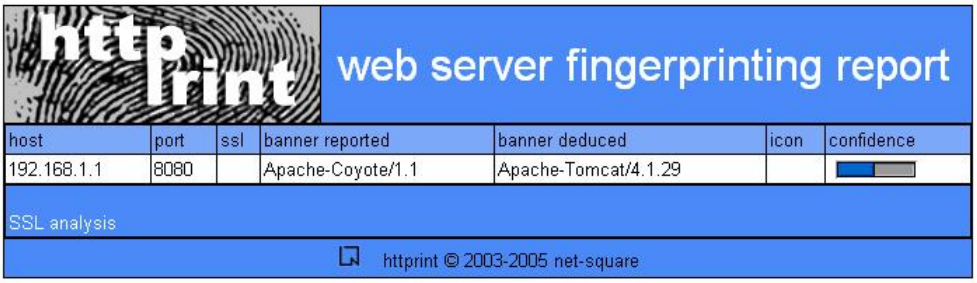

FIGURE 6.1 – Httpprint, analyse du fournisseur d'identité

Vulnérabilité Plusieurs détecteurs de vulnérabilité ont été utilisés en ce qui concerne le serveur WEB. Tout d'abord, metoscan pour quel type de requête est alloué sur le serveur :

```
bt ~ # metoscan http://192.168.1.1:8080

MetoScan - Simple HTTP Method Scanner

Method:                  GET     =>     200 (OK)
Method:                 POST     =>     200 (OK)
Method:                 HEAD     =>     200 (OK)
Method:                  PUT     =>     403 (FORBIDDEN)
Method:              OPTIONS     =>     200 (OK)
Method:               DELETE     =>     403 (FORBIDDEN)

--END--
```

Ensuite nikto en version http :

```
bt nikto # nikto.pl -host 192.168.1.1 -port 8080
---------------------------------------------------------------------------
- Nikto 2.02/2.03     -     cirt.net
+ Target IP:        192.168.1.1
+ Target Hostname: 192.168.1.1
+ Target Port:      8080
+ Start Time:       2008-11-05 15:26:05
---------------------------------------------------------------------------
```

```
+ Server: Apache-Coyote/1.1
- Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
+ OSVDB-397: HTTP method ('Allow' Header):
'PUT' method could allow clients to save files on the web server.
+ OSVDB-5646: HTTP method ('Allow' Header):
'DELETE' may allow clients to remove files on the web server.
+ OSVDB-877: HTTP method ('Allow' Header):
'TRACE' is typically only used for debugging and should be disabled.
This message does not mean it is vulnerable to XST.
+ OSVDB-39272: /favicon.ico file identifies this server as:
Apache Tomcat
+ OSVDB-0: GET / : Appears to be a default Apache Tomcat install.
+ OSVDB-6659: GET /ELDeadQuoFhinnhsYBEZ6ql3csanOOiN6JQRjei1cihetj3C
6IbxQImAfRByYX2kLpfsdzaH6xbrL2OHv749Y1yr3xUqOECjWTraYjLmlepOlHs9EBm
njHhdfKZHsCQreM1I6xAf4DRtF8bdLBpHNKiWlweKWIHyftvdnRiCGQSHN5DhNXCkyk
PtmXntqiBpe6bpoFV9yw47SCGROuoso1ayBQudW8C<font%20size=50>DEFACED
<!--//-- : MyWebServer 1.0.2 is vulnerable to HTML injection.
Upgrade to a later version.
+ 2967 items checked: 7 item(s) reported on remote host
+ End Time:        2008-11-05 15:26:14 (9 seconds)
---------------------------------------------------------------------
+ 1 host(s) tested
```

Et nikto en version https qui n'a pas abouti correctement :

```
bt nikto # nikto.pl -host 192.168.1.1 -port 8443
---------------------------------------------------------------------
- Nikto 2.02/2.03     -     cirt.net
+ Target IP:       192.168.1.1
+ Target Hostname: 192.168.1.1
+ Target Port:     8443
---------------------------------------------------------------------
+ SSL Info:        Ciphers: EDH-RSA-DES-CBC3-SHA
                   Info:    /CN=StyxIdP
                   Subject: /CN=StyxIdP
+ Start Time:      2008-11-05 15:27:30
---------------------------------------------------------------------
+ Server: Apache-Coyote/1.1
- Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
+ OSVDB-397: HTTP method ('Allow' Header):
'PUT' method could allow clients to save files on the web server.
+ OSVDB-5646: HTTP method ('Allow' Header):
'DELETE' may allow clients to remove files on the web server.
+ OSVDB-877: HTTP method ('Allow' Header):
'TRACE' is typically only used for debugging and should be disabled.
This message does not mean it is vulnerable to XST.
+ OSVDB-39272: /favicon.ico file identifies this server as:
Apache Tomcat
+ OSVDB-0: GET / : Appears to be a default Apache Tomcat install.
```

Nessus a aussi été employé, mais aucun résultat assez probant pour figurer dans ce

rapport a été obtenu. Aucune vulnérabilité pour la version 6.0.18 de Tomcat n'a été trouvé sur les sites suivants :
- http://nvd.nist.gov/
- http://www.securityfocus.com/
- http://www.frsirt.com/searchengine.php
- http://insecure.org/
- http://neworder.box.sk/
- http://www.milw0rm.com/search.php

6.3.3 Analyse de confiance

Adresse IP 192.168.1.1

Nom de la machine StyxIdP

Description de la confiance Le certificat du serveur est auto-signé et n'implique donc pas de tier pour vérifier le lien de confiance (chaîne de certification). La relation de confiance avec le fournisseur d'identitié est créée lors de la génération des cartes. Car une carte ne peut divulguer des informations seulement à son créateur (à cause de la double encryption de l'authentification forte). De plus le code PIN n'est jamais transmis au serveur et n'est stocké que dans la carte ce qui augmente la confiance.

6.3.4 Protection de la vie privée

Adresse IP 192.168.1.1

Nom de la machine StyxIdP

Police de protection de la vie privée Aucune information personnelle n'est transmise par le fournisseur d'identité à des tiers. La seule informations transmise par SAML est l'identifiant de l'utilisateur. Le nom et prénom qui sont stocké dans la carte ne sont jamais transmis à des tiers et son protégé par un code PIN personnel que seul l'utilisateur possède.

Violation de la vie privé Aucune atteinte à la vie privé n'a été trouvé dans le système (faille de sécurité permettant de recupérer des données). Néanmoins il faut que la LPD [1] soit respectée.
- Art 3 : Les données traitées sont des données personnelles. Ce qui nous oblige à avoir certaine précaution.
- Art 7 : Les données personnelles doivent protégées contre tout traitement non autorisé par des mesures organisationnelle (le compte administrateur) et des techniques approprié (l'application). Cet article est respecté.
- Art 7a : Si SAML transmet des données personnelles, l'utilisateur doit absolument être avertit de la transmission de ces données et de leur utilisation.
- Art 8 : L'utilisateur n'a pas directement le droit d'accès à ces données avec le logiciel existant, le maître doit lui donner ce droit d'accès sinon ce n'est pas légal.
- Art 11a : Le registre doit être déclaré auprès du préposé à la protection des données.
- Art 16-25 : traitement des données par des organes fédéraux. Ces articles sont importants à relever si un jour un système de ce type devait être mis en place.

1. RS35.1 loi fédérale sur la protection des données http://www.admin.ch/ch/f/rs/235_1/index.html

6.3.5 Résultats des tests

Les tests de l'OSSTMM ne s'appliquent pas vraiment pour sécuriser un prototype d'application web, car ils s'occupent plus généralement des failles du système d'exploitation ou des services mis en place sur le système tel que Tomcat. La sécurité de Tomcat a donc été évaluée même si cela n'est pas objectif car rien n'a été mis en oeuvre pour le protéger et pour protéger le serveur.

6.4 Test d'accès à l'Identity Provider (faille dans le code)

6.4.1 Principes

Le principe de ces tests se base sur le code source de l'application. Le but est de changer de point de vue et ne plus voir le code comme un moyen de réaliser une fonction, mais comme un moyen d'entrer ou de perturber l'application. Il est souvent possible d'obtenir le code d'une application assez facilement, par exemple par du "Social Engineering". Ou du décompilage de classes avec "Java Decompiler [2]" pour java par exemple.
Le but est donc de trouver pour chaque partie de l'application les failles qui sont visibles en utilisant son code source. Pour le fournisseur d'identité voici les grandes parties analysées : servlet d'administration, servlet d'authentification, pages du dossier "management", et pages directement visibles à la racine.

6.4.2 Servlet d'administration

Faits Le servlet d'administration ne doit normalement pas être visible pour le commun des utilisateurs mais seulement par l'administration.

Tests

- https://styxidp:8443/StyxWebSecurity_IdP/Management/admin?type=getIdPPK
  On obtient la clé public ce n'est pas grave.
- https://styxidp:8443/StyxWebSecurity_IdP/Management/admin?type=getCMKeys
  On obtient les clés du CardManager des cartes c'est très grave. Voici les données obtenues :
  `@ABCDEFGHIJKLMNO @ABCDEFGHIJKLMNO @ABCDEFGHIJKLMNO`
- https://styxidp:8443/StyxWebSecurity_IdP/Management/admin?type=genStoCert&eid=3&efn=firstname&eln=lastname
  On pourrait en ayant le code source et un ID utilisateur déjà inscrit modifier son certificat et donc l'empêcher de se loguer

Résultats Il faut donc absoluement resteindre l'accès au servlet en local seulement ou avec l'ip d'administration dans le web.xml

6.4.3 Servlet d'authentificaton

Faits Le servlet d'authentification ne doit être accessible que par l'applet WEB d'authentification.

Tests

- https://styxidp:8443/StyxWebSecurity_IdP/auth?type=request_uc&eid=1
  Permet de voir un challenge crypté dont on ne connait pas l'orginal, ce n'est pas un risque de sécurité. On connait par contre l'utilisateur et on peut donc voir si un utilisateur avec l'ID en paramètre existe ce qui peut s'avèrer dangereux. Par exemple avec : https://styxidp:8443/StyxWebSecurity_IdP/auth?type=request_uc&eid=1. On pourrait donc savoir combien d'utilisateur sont inscrit dans l'identity provider. Challenge de l'utilisateur inscrit :

2. http://java.decompiler.free.fr/

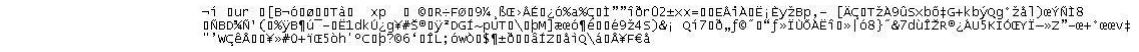

FIGURE 6.2 – Servlet d'authentification, challenge plein

Challenge d'un utilisateur non inscrit :

FIGURE 6.3 – Servlet d'authentification, challenge vide

Il est donc très facile de connaître l'inscription d'un utilisateur (identitfiant) dans le système.

- https://styxidp:8443/StyxWebSecurity_IdP/auth?type=openSPconnection
  Permet de savoir, s'il on a le cookie du HTTPSession si un utilisateur possède une requête d'authentification sur le serveur d'identification à traiter (envoyé réponse à un service).
- https://styxidp:8443/StyxWebSecurity_IdP/auth?type=response_&eid=1
  Un hacker peut tenter sa chance mais pour qu'il crypte le bon challenge et l'envoi en post avec cette commande, les chances sont nulles.
- https://styxidp:8443/StyxWebSecurity_IdP/auth?type=logout&eid=1
  Grande faille de sécurité, un petit script qui déconnecte tous les utilisateurs est très facile à faire, il suffit d'incrémenter le eid. Bien entendu cette fonction est en libre accès. Mais elle déconnecte l'utilisateur qui a la session en cours et pas celui avec le eid précisé.

Résultats Il faut absolument sécuriser l'accès à cette applet ou au moins à certaines fonctions pour empêcher la déconnexion sauvage d'utilisateur ou l'analyse d'information tel que l'existence d'un utilisateur pour un id.

6.4.4 Servlet de téléchargement

Faits Le servlet de téléchargement doit être accessible que par l'administrateur sauf le fichier de métadonnées qui est en libre accès.

Tests

- https://styxidp:8443/StyxWebSecurity_IdP/Management/download?type=meta
  Ok on obtient le fichier de métadonnées mais il doit être en libre accès donc ce n'est pas une faille de sécurité.
- https://styxidp:8443/StyxWebSecurity_IdP/Management/download?type=pri&name=StyxSPBankDemo
  Accès protégé.
- https://styxidp:8443/StyxWebSecurity_IdP/Management/download?type=pub&name=StyxSPBankDemo
  Accès protégé.
- https://styxidp:8443/StyxWebSecurity_IdP/Management/download?type=cer&name=StyxSPBankDemo
  Accès protégé.
- https://styxidp:8443/StyxWebSecurity_IdP/Management/download?type=downloadcert
  Accès protégé.

Résultats Le servlet est sécurisé correctement.

6.4.5 Pages de Management

Totes les pages sont bien protégées et seulement accessible en local ou avec `admin_ip` du `web.xml`.

6.4.6 Pages accessibles

Faits Les pages ne doivent pas divulguer d'élément d'erreur ou permettre de faire des opérations nuisibles avec le logiciel.

Tests

- https://styxidp:8443/StyxWebSecurity_IdP/admin.jsp
  inaccessible, accès non autorisé
- https://styxidp:8443/StyxWebSecurity_IdP/check_auth.jsp
  Permet de savoir si l'on possède le cookie correspondant à l'utilisateur s'il est authentifié ou non au fournisseur d'identité.
- https://styxidp:8443/StyxWebSecurity_IdP/login_control.jsp
  Ne permet pas d'effectuer des operations nuisibles.
- https://styxidp:8443/StyxWebSecurity_IdP/login.jsp
  Ne permet pas d'effectuer des operations nuisibles.
- https://styxidp:8443/StyxWebSecurity_IdP/saml_auth.jsp
  Ne permet pas d'effectuer des operations nuisibles.
- https://styxidp:8443/StyxWebSecurity_IdP/recv_saml_auth_request.jsp
  Ne permet pas d'effectuer des opérations nuisibles mais permet de voir une exception qui indique quel framework est utilisé ce qui pourrait aider des personnes malveillantes à trouver des failles.
  `org.opensaml.saml2.binding.`
  `decoding.HTTPRedirectDeflateDecoder.doDecode(`
  `HTTPRedirectDeflateDecoder.java:96)`
  `org.opensaml.ws.message.decoder.BaseMessageDecoder.decode(`
  `BaseMessageDecoder.java:71)`
  `net.clareitysecurity.websso.idp.HttpHandler.decodeSAMLRequest(`
  `HttpHandler.java:114)`
- https://styxidp:8443/StyxWebSecurity_IdP/admin.jsp
  Il faut déjà pouvoir y accéder mais aucune erreur SQL n'est retournée, il n'est donc pas possible de faire du SQL injection facilement.

Résultats Il faut traiter les exceptions qui remonte car elles divulguent des informations sur les framework utilisés et permettent peut-être à une personne malveillante d'utiliser les failles de ses frameworks à l'insu du logiciel.

6.4.7 Résultat des tests

Plusieurs failles de sécurité ont été relevées par ces tests. Ces failles permettent d'obtenir des informations utiles à des personnes malveillantes, mais aussi de paralyser le système ce qui pose vraiment un problème. L'accès aux différents servlets doit donc être plus sécurisé.

6.4.8 Corrections

Les corrections des principales failles de sécurité ont été appliqué à la version 1.2.

- Sécurisation du servlet d'administration.
- Sécurisation des exceptions des pages JSP.
- Sécurisation du servlet d'authentification sauf l'obtention du challenge crypté qui nécessite la mise en place de l'authentification de l'applet (par exemple par un challenge donné au chargement de l'applet pour l'authentifier lors de l'appel de méthode). Il est aussi important de préciser qu'après un examen plus approfondi, la faille de la fonction de logout était un faux-positif.

6.5 Visualisations des données échangées

Le sniffage du réseau s'est déroulé à l'aide du logiciel WireShark sur le PC client. Voici un rappel de la structure du réseau privé relatifs aux tests :

- StyxIdP, 192.168.1.1, Identity Provider Tomcat Server
- StyxSPBankDemo, 192.168.1.2, Service Provider Tomcat Server
- StyxClient, 192.168.1.3, Client qui se connecte à l'aide du navigateur WEB

Trois cas de figure sont essentiellement traités dans ce sous-chapitre, ce sont les plus importants car les données transites par des connexions non-sécurisées (Internet). Les données qui transite sur le fournisseur d'identité uniquement sont beaucoup moins vulnérable et ne sont pas traitées dans ces tests.

6.5.1 Authentification à l'Identity Provider

Etat avant la mesure La page `login.jsp` et son applet sont chargé dans le navigateur web. On se connecte sur la carte avec le PIN aucune transmission n'est effectuée jusqu'au clique sur continuer.

Déroulement

1. Applet WEB vers IdP : appelle du challenge
2. IdP vers Applet WEB : envoi le challenge
3. Applet WEB vers IdP : Envoi la réponse

Résultat On ne voit aucune information, tout est crypté, on voit juste les trames SSL de version 3. Dans la mesure en annexe `idp_authentification.pcap`, les trames 12, 15,15,16,19,23,27 et 27 transporte des données cryptées. En voici un exemple :

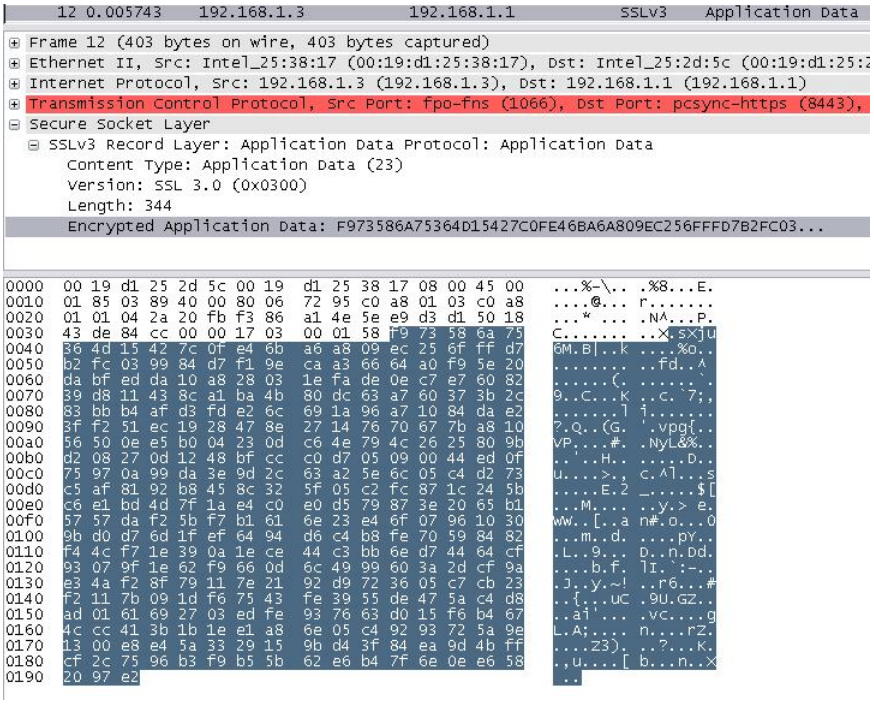
12 0.005743 192.168.1.3 192.168.1.1 SSLv3 Application Data
Frame 12 (403 bytes on wire, 403 bytes captured)
Ethernet II, Src: Intel_25:38:17 (00:19:d1:25:38:17), Dst: Intel_25:2d:5c (00:19:d1:25:
Internet Protocol, Src: 192.168.1.3 (192.168.1.3), Dst: 192.168.1.1 (192.168.1.1)
Transmission Control Protocol, Src Port: fpo-fns (1066), Dst Port: pcsync-https (8443),
Secure Socket Layer
SSLv3 Record Layer: Application Data Protocol: Application Data
Content Type: Application Data (23)
Version: SSL 3.0 (0x0300)
Length: 344
Encrypted Application Data: F973586A75364D15427C0FE46BA6A809EC256FFFD7B2FC03...

FIGURE 6.4 – Exemple de trame de données SSL

6.5.2 Envoi d'une requête SAML depuis le SP à l'IdP

Etat avant la mesure La page `index.html` est chargée au départ.

Déroulement

1. SP : Chargement de la page de login.jsp
2. SP : Redirection vers la page d'envoi de requête
3. SP : Redirection de la requête SAML vers le Service Provider
4. IdP : réception de la requête pas `recv_saml_auth_request.jsp`
5. IdP : Affichage de la page de login avec l'applet WEB.

Résultat La mesure est disponible en annexe `send_saml_request.pcap`. Le navigateur demande d'abord la page `login.jsp` dans la trame 8, ensuite il est redirigé avec la page `request_auth.jsp` puis encore une fois redirigé avec la trame 12 vers la version https de la page précédente. Donc une connexion SSL s'ouvre entre le client et le SP. Depuis lors aucune des données n'est visible (trame 24). Une autre connexion SSL s'ouvre depuis le client vers l'IdP lorsque que la requête est redirigé mais rien n'est visible en clair (trame 38, 41). Les dernière connexion SSL servent à transporter la page de login de l'IdP et son applet Java.

6.5.3 Envoi d'une réponse avec authentification depuis l'IdP au SP

Etat avant la mesure La page login.jsp et son applet sont chargés dans le navigateur web. On se connecte sur la carte avec le PIN aucune transmission n'est effectuée jusqu'au clique sur continuer.

Déroulement

1. Applet WEB vers IdP : appelle du challenge
2. IdP vers Applet WEB : envoi le challenge
3. Applet WEB vers IdP : Envoi la réponse
4. IdP : `saml_auth.jsp` renvoi la réponse au SP
5. SP : `recv_saml.jsp` reçoit la réponse et affiche la page des cadres
   Attention, durant cette étape le fichier de méta-données est téléchargé pour contrôler la signature de la réponse

Résultat La mesure se trouve dans le fichier `send_saml_response_and_service_access.pcap`. Appel du challenge commence à la trame 6. Le challenge est transféré à partir de la trame 15 et 20, 21 ,22, 23. Ensuite la réponse au challenge est transmise depuis l'applet WEB au serveur d'identité. En voit à partir de la trame 37 une communication entre le poste client et le fournisseur de service, l'accès à la page `recv_saml.jsp` est donc clair. En voit le téléchargement non protégé du fichier de métadonnée ce qui est normal (trame 53, 54 et 55). La suite des trames est utilisé pour le téléchargement des pages du fournisseur de service ainsi que pour les challenge-response échangés durant l'authentification.

6.5.4 Résultats

Les tests ont prouvés que toutes les données sensibles et confidentielles sont transférées à travers des tunnel SSL qui permettent de sécuriser les transactions. Aucune donnée confidentielle n'est donc directement visible, ce qui assure une plus grande sécurité de l'infrastructure.

Conclusion

Tous ces tests et leurs analyse ont permis de mettre à jour plusieurs problèmes et de les résoudre dans des nouvelles versions. Aucun grand problème de conception ou de spécification a été relevé et l'infrastructure globale parait cohérente sur la partie testée qui est le fournisseur d'identité. Plusieurs autres parties dont le fournisseur de service n'ont pas été testées. Le fournisseur de service n'est pas optimal et ne peut pas être testé dans son état actuel à cause du système de gestion de la pérennité de l'authentification entre lui et le fournisseur d'identité. Actuellement c'est un système haut-niveau en Java script pour les besoins de la démonstration qui est utilisé, mais pour une sécurité accrue, il faudrait utilisé des services WEB pour que les 2 fournisseurs communiquent directement entre eux.

7 Déploiement

Introduction

Ce chapitre explique comment générer les fichiers exécutables et package pour distribuer l'application, comment installer un nouveau système depuis les packages générés et comment installer les sources dans les environnement de développement.

7.1 Compilation et assemblage

Les fichiers sont auto-compilé avec Eclipse. Il faut bien entendu vérifier qu'ils soient tous compiler avant des créer les packages.

Applet NXP JCOP Card Le fichier binaire important à mettre de côté est `StyxWebSecurity.cap`. Il contient l'application qui doit être installé sur la carte. Il se trouve dans le projet Eclipse, plus précisément dans le dossier `bin\StyxWebSecurity\javacard\`.

Applet WEB d'administration Le fichier généré est un jar qui doit seulement comprendre les fichiers suivants :

- `META-INF\MANIFEST.MF` Il faut ajouter cette ligne `Class-Path: rmioffcard.jar offcard.jar`
- `StyxWebSecurity\AdminApplet\StyxAppletAdmin.class` GUI de l'applet
- `StyxWebSecurity\AdminApplet\StyxIDPConnector.class` Connecteur vers l'Identity Provider
- `StyxWebSecurity\AdminApplet\StyxJCConnector.class` Connecteur vers la JavaCard
- `StyxWebSecurity\AdminApplet\StyxNewCardProcess.class` Processus de création d'une nouvelle carte
- `StyxWebSecurity\AdminApplet\StyxStartActionListenner.class` Action qui lance le processus
- `StyxWebSecurity\StyxJCInterface.class` Interface pour RMI et la JavaCard
- `jct.dll` Pilote obligatoire
- `StyxWebSecurity.cap` Fichier de l'application on-card disponible dans le projet JavaCard

Avec Eclipse, cliquez sur le projet avec le bouton droit, puis "'export to jar file".

Applet WEB d'authentification Le fichier généré est un jar qui doit seulement comprendre les fichiers suivants :

- `META-INF\MANIFEST.MF` Il faut ajouter cette ligne `Class-Path: rmioffcard.jar offcard.jar`
- `StyxWebSecurity\AuthApplet\StyxAppletAuth.class` GUI de l'applet
- `StyxWebSecurity\AuthApplet\StyxAuthProcess.class` Processus d'authentification
- `StyxWebSecurity\AuthApplet\StyxContinueActionListenner.class` Action qui contine l'authentification
- `StyxWebSecurity\AuthApplet\StyxIDPConnector.class` Connecteur vers l'Identity Provider
- `StyxWebSecurity\AuthApplet\StyxJCConnector.class` Connecteur vers la JavaCard
- `StyxWebSecurity\AuthApplet\StyxModifyActionListenner.class` Action qui modifie PIN
- `StyxWebSecurity\AuthApplet\StyxStartActionListenner.class` Action qui lance l'authentification
- `StyxWebSecurity\StyxJCInterface.class` Interface pour RMI et la JavaCard
- `jct.dll` Pilote obligatoire

Avec Eclipse, cliquez sur le projet avec le bouton droit, puis "'export to jar file".

Base de données Il faut générer le fichier SQL pour recréer les tables. On peut le générer avec "MySQL Administrator" Il faut bien entendu ne pas reprendre les données déjà insérées mais juste la structure des tables et les procédures stockées.

Service Provider et Identity Provider Il faut cliquer sur le projet Eclipse et exporter sous la forme d'archive war.

7.2 Installation et configuration

7.2.1 Installation d'un fournisseur d'identité

Il faut d'abord installer Apache Tomcat et MySQL sur le serveur d'identité. Ces 2 logiciels sont présent sur le CD ou disponible sur leur site Internet respectif

Mysql et configuration

Voici les principales étapes :

1. Créer un utilisateur local
2. Il faut bloquer les accès au utilisateur local seulement
3. Il suffit d'exécuter le script d'installation SQL

Tomcat et configuration

Voici les principales étapes :

1. Ajoutez les libraires endorsed à la base du dossier tomcat dans un dossier endorsed.
2. Déployez le war depuis le Tomcat Manager.
3. Modifiez le web.xml de l'application déploiée `\webapps\StyxWebSecurity_IdP\WEB-INF\web.xml`
   - `dataBaseURL` devrait être le même car il est conseillé d'installer MySQL en local
   - `dataBaseUser` selon configuration MySQL
   - `dataBasePassword` selon configuration MySQL
   - `IdPName` nom de la machine de l'Identity Provider
   - `IdPCheckAuth` modifier le nom de la machine
   - `IdPRecvSAMLURL` modifier le nom de la machine
   - `admin_ip` IP supplémentaire pour l'accès à l'administration
   - Modifier `<transport-guarantee>CONFIDENTIAL</transport-guarantee>` en `<transport-guarantee>NONE</transport-guarantee>`
4. N'oubliez pas de relancer le serveur.
5. Allez dans l'administration du serveur sans SSL "admin" et "styx", sous "Provider" et "KeyTool : installation of keystore", créer en un nouveau.
6. Re-modifier le `web.xml` : `NONE` redevient `CONFIDENTIAL`.
7. Copier les applet WEB `StyxWebSecurityAdminApplet.jar` et `StyxWebSecurityAuthApplet.jar` dans le dossier du serveur Tomcat `\webapps\StyxWebSecurity_IdP\Applets`.
8. Signer les fichiers précédent avec `jarsigner nomfichier.jar styxidp`.
9. Modifier le fichier `server.xml` pour ajouter le SSL. Plus de détails sont fournis dans le chapitre "Implémentation", "Sécurisation des fournisseurs d'identité et de service".
10. Ne pas oubliez d'installer le pilote Windows lors du branchement du lecteur de carte SCR3310.

7.2.2 Installation d'un fournisseur de service

Il faut d'abord installer Tomcat sur la machine qui fournit le service.

Tomcat et configuration

Voici ensuite les étapes de configuration :

1. Ajoutez les libraires endorsed à la base du dossier tomcat dans un dossier endorsed.
2. Déployez le war depuis le Tomcat Manager.
3. Modifiez le web.xml de l'application déploiée
   `\webapps\StyxServiceProvider\WEB-INF\web.xml`
   - `privatekeyfile` emplacement de la clé publique
   - `publickeyfile` emplacement de la clé privée
   - `IdPMetadataFileURL` modifier nom de la machine de l'Identity Provider
   - `IdPLoginControlURL` modifier le nom de la machine de l'Identity Provider
   - `IdPRecvSAMLURL` modifier le nom de la machine du Service Provider
   - `IdPName` modifier le nom de la machine de l'Identity Provider
   - `SPRecvSAMLURL` modifier le nom de la machine du Service Provider
   - `SPName` modifier le nom du Service Provider
4. Utiliser `install_keys.jsp` pour installer les clés dans le keystore et pour utiliser SSL.
5. Modifier le fichier `server.xml` pour ajouter le SSL. Plus de détails sont fournis dans le chapitre "Implémentation", "Sécurisation des fournisseurs d'identité et de service".
6. N'oubliez pas de relancer le serveur.

7.2.3 Installation du client

Le client doit obligatoirement avoir installer le JDK et JRE 1.5 sur son poste. C'est aussi la version de Java qui doit être employé avec Internet Explorer et Firefox. Ne pas oubliez d'installer le pilote Windows lors du branchement du lecteur de carte SCR3310.

7.3 Mise en place des environnements de développement

Il est nécessaire d'installer au moins 3 versions d'Eclipse :

1. Eclipse avec plugin JCOP : Pour l'applet NXP JCOP et pour les 2 applets WEB (StyxWebSecurityAdminApplet et StyxWebSecurityAuthApplet).
2. Eclipse Java EE : Pour le Service Provider.
3. Eclipse Java EE : Pour l'Identity Provider.

Il faut ensuite importer les projets correspondants dans chaque une des versions :

1. Dans le "Project Explorer", clique droit puis "Import", enfin "Import...".
2. Sous "General", "Existing Projects into Workspace".
3. Ensuite "Next".
4. Sous "Select root directory", indiquez le chemin du dossier projet Eclipse dans les sources.
5. Ensuite "Next", le projet est finalement importé.
6. En ce qui concerne les projets des Services Provider et Identity Provider, il faut configurer les serveurs Tomcat (onglet "server", clique droit, "new", indiquer la version et l'emplacement dans "Server Runtime"),et la configuration du server.xml avec SSL pour que ils soient réellement actifs. Plus de détails sont fournis dans le chapitre "Implémentation", "Sécurisation des fournisseurs d'identité et de service".

7.4 Les logiciels

7.4.1 Eclipse et JCOP

Deux versions d'Eclipse ont été employées pour ce projet :
- Eclipse avec JCOP : fourni sur le cd livré avec les cartes NXP.
- Eclipse Java EE : téléchargé sur le site http://www.eclipse.org/downloads/

Installation Voici la procédure d'installation à suivre :

1. Installez le Sun JDK 1.4 avec le JRE dans `c:\jdk14`
2. Installez le Sun JDK 1.5 avec le JRE dans `c:\jdk15`
3. Installez le pilote SCR3310 présent sur le cd.
4. Décompressez Eclipse dans un dossier de votre choix.
5. Créez un raccourci pour lancer eclipse avec le JRE 1.4, par exemple : `D:\eclipse\eclipse.exe -vm "c:\jdk14\jre\bin\javaw.exe"`
6. Lancez Eclipse et dans "Help/Software Update/Find and install", sélectionnez "Search for new features to install" et cliquez sur "next".
7. Décompressez JCOP Tools et PH Targets Pack dans des dossiers temporaires.
8. Cliquez sur "New local site" pour ajouter un site correspondant au dossier décompressé de JCOP Tools et PH TargetPack.
9. Accepter les certificats durant l'installation.

Lors de la création d'un projet, insérer une carte NXP JCOP pour activer le plugin.

7.4.2 JavaScript Form Validation

Pour valider les formulaires avant de les soumettre, la bibliothèque "JavaScript Form Validation" a été utilisée. Elle permet facilement de contrôler :
- le type de valeur : alpha, numeric, alphanumeric, withspace, withoutspace, email
- la longueur max.
- la longueur min.
- les champs requis.

Pour l'utiliser, il faut copier le fichier `gen_validatorv31.js` et inséré ce code dans la balise `<head>` :

```
<script language="JavaScript" src="gen_validatorv31.js"
type="text/javascript"/>
```

Ensuite pour chaque formulaire, il faut déclarer un "formvalidator" :

```
<SCRIPT language="JavaScript">
 var frmvalidator  = new Validator("FORMNAME");
 frmvalidator.EnableOnPageErrorDisplaySingleBox();
 frmvalidator.EnableMsgsTogether();
 frmvalidator.addValidation("FIELDNAME","req","ERRORMSG");
 frmvalidator.addValidation("FIELDNAME","maxlen=60","ERRORMSG");
 </script>
```

Cette balise est utilisée pour préciser où les messages d'erreur sont affichés :

```
<div id='FORNAME_errorloc' class='error_strings'></div>
```

Plus de précisions peuvent être trouvées sur le site de l'éditeur :
http://www.javascript-coder.com/html-form/javascript-form-validation.phtml.

7.4.3 Jarsigner

Cet outils est fournis avec la plateforme Java et permet de signer des archives JAR et par conséquent des applets WEB. Il est utilisé dans le projet pour signer les archives des applets d'administration et d'authentification ainsi que les archives des API d'accès à la JavaCard. Pour signer un JAR, il faut préalablement avoir une identité dans le keytool. Ensuite un peu signer une archive avec cette commande :

```
jarsigner jar-file.jar alias_in_keystore
```

On peut aussi vérifier une signature :

```
jarsigner -verify jar-file.jar
```

Des informations plus précises sur cet outil sont disponible à cette adresse :
http://java.sun.com/j2se/1.3/docs/tooldocs/win32/jarsigner.html

7.4.4 Keytool

Le keytool est un outil de la plateforme Java qui permet de gérer des identités. Toutes les identités sont sauvées par défaut dans le keytsore (fichier `.keytsore` qui se trouve dans le dossier de l'utilisateur courant).

Génération d'une nouvelle identité la génération d'une nouvelle identité se fait avec la commande :

```
keytool -genkey -alias duke -keypass dukekeypasswd
```

Ensuite il faut répondre au différentes questions pour générer le certificat. Attention les clés RSA ont une longueur maximal de 1024bit.

```
Tapez le mot de passe du keystore: *****
Quels sont vos prénom et nom ?
[Unknown]: David Olivier
Quel est le nom de votre unité organisationnelle ?
[Unknown]: Java Card Fed Man
Quel est le nom de votre organisation ?
[Unknown]: EIF
Quel est le nom de votre ville de résidence ?
[Unknown]: Fribourg
Quel est le nom de votre état ou provinc ?
[Unknown]: Fribourg
Quel est le code du pays à deux lettres pour cette unité ?
[Unknown]: CH
Est-ce CN=David Olivier, OU=Java Card Fed Man, O=EIF,
                          L=Fribourg, ST=Fribourg, C=CH ?
[non]:oui
```

Liste des identités présentes dans le keystore Il suffit de taper la commande suivante et d'entrer le mot de passe du keystore :

```
keytool -list
```

Voici un exemple d'affichage :

```
Type Keystore : jks
Fournisseur Keystore : SUN

Votre Keystore contient 1 entrée(s)

mike, 28 oct. 2008, keyEntry,
Empreinte du certificat (MD5) :
             AA:88:A7:45:E0:00:0B:1A:17:A1:F0:AC:99:87:6D:37
```

Conclusion

L'installation du logiciel en soit est assez simple. Ce qui est complexe c'est s'imaginer l'infrastructure complète du prototype, pour cela le diagramme de déploiement dans la partie spécification peut être utile. Le développement ou le déploiement sur d'autre système d'exploitation n'ont pas été testés mais sont tout a fait envisageable car seul Java est utilisé comme langage de programmation et il est multi-plateforme. Toute modification d'un composant remet en cause tous les composants du système car certaines fonctions sont complémentaires. C'est pour cela qu'il est recommandé de comprendre complètement l'infrastructure avant toute modification.

8 Conclusion

Le projet Secure Java Card for Federate Identity Management a atteint les objectifs établis dans le cahier des charges et dans les délais impartis. Le résultat est tout d'abord une analyse détaillée des principales technologies de Federate Identity Management présentes sur le marché. Cette analyse à permis de choisir SAML pour son ouverture et sa sécurité élevée afin de concevoir un prototype. Lui-même complètement documenté et fonctionnel, il démontre son utilisation. L'analyse a aussi permis de placer la JavaCard pour sécuriser le prototype. L'authentification forte est le concept qui a été retenu et mis en place à l'aide des cartes JCOP et de l'algorithme à clé asymétrique RSA ainsi que du concept de "challenge/response". La principale innovation est l'utilisation des cartes à puce JavaCard à travers le navigateur Internet de l'utilisateur.

Ce projet a été passionnant. Il a permis de découvrir un environnement de développement JCOP sans conteste totalement différent du simulateur JCWDE, avec ses multiples contraintes : taille maximale des tableaux sur RMI, garbage collector. Le point le plus critique était clairement l'architecture très complexe imposée par tous les composants de différente nature. Une remarque est tout de même à faire sur ma déception d'un projet géré complètement seul et qui n'habitue pas les gens à collaborer entre eux. Bien entendu si le projet avait été fait par 2 personnes, le prototype aurait peut-être pu donner un produit.
Un autre élément captivant dans le projet était l'aspect sécuritaire : Où placer la JavaCard pour rendre plus imperméable le système ? Comment sécuriser une carte ? Comment procéder au tests ? Quel algorithme utiliser et pourquoi ? Toutes ses questions ne se posent pas dans tous les projet de diplôme et m'ont permis d'accroître mon expérience dans ce domaine.
Un autre élément intéressant était le développement d'application riche WEB à travers des applets. Il est très utile pour accéder au ressource d'un PC client depuis une page Internet et dans le projet nous offre la possibilité d'accéder au lecteur de carte du client et par conséquent à sa carte pour établir l'authentification.

Finalisation du prototype Pour obtenir relativement rapidement un produit avec le prototype existant, il faut rajouter ou modifier des fonctionnalités. Premièrement il faut utiliser des services WEB pour que les fournisseur d'identité puissent communiquer directement avec les fournisseur de service et vice-versa. Dans un deuxième temps, on devrait modifier le système de déconnexion des utilisateurs en utilisant des services WEB et pas de Java script non sécurisé comme actuellement. L'échange d'information sur les utilisateurs est importante c'est pour cela qu'il faudrait permettre au fournisseur de demander certaines informations en plus de l'authentification au fournisseur de service, en obligeant bien entendu l'utilisateur à autoriser ce transfert. Le bug de l'exposant fixe des clés générées dans la plateforme JCOP doit absolument être corrigé pour des raisons évidentes de sécurité. Et finalement il faudrait aussi protéger l'applet WEB d'authentification et ses communications avec le serveur en l'identifiant par exemple avec un challenge.

Perspectives économique L'identification numérique des personnes à travers Internet est pour l'instant peu sécurisée et rébarbative. Les utilisateurs on des centaines d'identifiants et de mots de passe différents ce qui rend l'identification compliquée et peu fiable. SAML apporte une ouverture car il permet de n'avoir qu'une identité qu'on peut utiliser avec tous les services. D'un autre côté la sécurité des identités sur Internet est très peu fiable. Dans la plupart des cas elles sont même facilement usurpées. Le concept d'authentification forte avec quelque chose que l'utilisateur possède, permet de sécuriser plus fortement l'authentification. La technologie JavaCard dans ce projet à donc été très utile pour renforcer la cohésion du système - fournir une plateforme de gestion d'identité et fournir une authentification forte.

Perspectives de développement Les perspectives d'avenir pour ce projet sont assez conséquentes car le marché ne dispose pas encore de grand monopole dans ce domaine et aucun produit qui mélange ces 2 technologies dans le but de la SSO est disponible. Par contre plusieurs produits soit pour l'authentification forte (JavaCard et WEB) soit pour les FIM (SAML) existent.
Bien entendu le prototype pourrait d'abord être enrichi de quelques fonctionnalités comme par exemple : Offrir la possibilité au utilisateurs de modifier leurs informations personnelles à travers une interface de gestion de leur compte. Ou ajouter la notion de groupe d'utilisateur, permettant de regrouper les utilisateurs ayant les même droits. Et même ajouter les gestions des droits par fournisseur de service. Par exemple tel groupe ou tel utilisateur à le droit de se connecter. Et aussi transférer et utiliser les droits (groupes) du côté du fournisseur de service. Mais le plus important reste l'ouverture sur l'utilisation des JavaCard pour par exemple stocker et utiliser les certificats à l'intérieur d'une carte, cela incluant la validation de la chaîne de certification et l'extraction de leur clé publique.

9 Remerciements

Je remercie toutes les personnes ayant contribué au projet, plus particulièrement :
- Les professeurs : M. Joye et M. Scheurer, pour l'accompagnement du projet et leurs conseils et leurs implications.
- Le responsable externe : M. Weissbaum, pour ses conseils avisés.
- Les collaborateurs techniques : M. Roche et M. Nguyen, pour le support logistique (matériel, salle)

10 Figures

Ce chapitre recense toutes les images et schémas présents dans le document. Ils sont répertoriés dans l'ordre de leur apparition.

3.1 SSO 10
3.2 Identification OpenID 11
3.3 Identification SAML 14
3.4 OpenID Authentification with esthablished Association, J.Hodges www.identitymeme.org 20
3.5 SAML HTTP Redirect or POST based Web Browser SSO Profile, J.Hodges identitymeme.org 21
3.6 OpenID Authentification without Established Association, J.Hodges identitymeme.org 22
3.7 SAML Web Browser SSO Flow with Artifact Binding used on Reply form IdP, J.Hodges identitymeme.org 23
3.8 SAML Web Browser SSO Profile "Unsolicited Response", J.Hodges identitymeme.org 24
3.9 Java Card Technology : Strong Authentication, Cryptographic Authentication with Java Card Technology Ellen Siegel and Matt Hill, JavaOne Sun Session TS-5251 25
3.10 Architecture avec gestion centrale 28
3.11 Architecture dé-centralisée 29
3.12 Signature du certificat depuis la carte 29
3.13 Processus d'authentification avec le certificat signé sur la carte 30
3.14 Architecture globale pour OpenID ou SAML 32
3.15 Architecture pour "Java Card Safe" 34

4.1 Cas d'utilisations pour l'application on-card 38
4.2 Classes pour l'application on-card 40
4.3 Cas d'utilisations pour les applets 41
4.4 Composants de communication, applet d'administration 42
4.5 Séquence génération de carte, applet d'authentification 43
4.6 Classes pour applet d'administration 44
4.7 Cas d'utilisations pour les applets 45
4.8 Composants de communication, applet d'authentification 46
4.9 Séquence authentification, applet d'authentification 46
4.10 Classes pour applet d'authentification 48
4.11 Diagramme d'état pour le processus d'authentification 49
4.12 Cas d'utilisations pour les applets 50
4.13 SAML, Communication du côté du fournisseur d'identité 53
4.14 Diagramme de classe des servlets 54
4.15 Diagramme de classe de la logique métier de l'identity provider 55
4.16 SAML, Processus de traitement du côté du fournisseur de service 58
4.17 Déploiement des composants 59

5.1 Applet d'administration, implémentation, demande du pin 67

5.2 Applet d'administration, implémentation, carte en cours de création 67
5.3 Applet d'administration, implémentation, création de la carte terminée 68
5.4 Applet d'authentification, implémentation, demande du pin 68
5.5 Applet d'authentification, implémentation, auhtentifié à la carte mais pas encore à l'Identity Provider 69
5.6 Applet d'authentification, implémentation, authentifié à l'Identity Provider . 69
5.7 Applet d'authentification, implémentation, déconnecté à cause du retrait de la carte 69
5.8 Identity Provider, implémentation, page d'accès à l'espace d'administration . 75
5.9 Identity Provider, implémentation, page principale de l'espace d'administration 75
5.10 Identity Provider, implémentation, page d'ajout de nouvel utilisateur 76
5.11 Identity Provider, implémentation, page de modification d'un utilisateur . . . 77
5.12 Identity Provider, implémentation, page de visualisation d'utilisateur 78
5.13 Identity Provider, implémentation, page de visualisation des Service Providers 78
5.14 Service Provider, implémentation, page d'accueil 86
5.15 Service Provider, implémentation, page protégée 87
5.16 Service Provider, implémentation, page de déconnexion 87

6.1 Httpprint, analyse du fournisseur d'identité 117
6.2 Servlet d'authentification, challenge plein 122
6.3 Servlet d'authentification, challenge vide 122
6.4 Exemple de trame de données SSL 126

11 Bibliographie

11.1 Livres, revues, rapports

IEEE WETICE05, Pluggin a Scalable Authentication Framework into Shibboleth
N. Zhang, L. Yao, J. Chin, Q.Shi, A. Nenadic, A. McNab, A. Rector and C.Goble. University of Manchester and Liverpool John Moores University, 2005.

John Wiley and Sons, Achieving fine-grained access control in virtual organizations
N. Zhang, L. Yao, A. Nenadic, J. Chin, C. Goble, A. Rector, D. Chadwick, S. Otenko and Q. Shi. University of Manchester, University of Kent, Liverpool John Mores University, 2006.

IEEE Security and Privacy, Identity Management
March/April 2008, Volume 6, Number 2

Java Card Security
Projet de semestre 2008, Aeschlimann Johann et Olivier David, EIA-FR

OpenID 2.0 : A Platform for User-Centric Identity Management
David Recordon VeriSign Inc et Drummond Reed Cordance Corporation

Next Steps for Security Assertion Markup Language(SAML)
Samir Saklikar and Subir Saha Motorola India Research Labs

11.2 Sites WEB

Apache Tomcat SSL Howto
http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html
http://tomcat.apache.org/tomcat-3.3-doc/tomcat-ssl-howto.html
http://java4it.blogspot.com/2007/01/how-to-configurer-tomcat-avec-https.html

Applet and Server Communication
http://www.j-nine.com/pubs/applet2servlet/Applet2Servlet.html
http://manu.e3b.org/Java/Tutoriels/ProgrammationReseau/AppletServlet.htm

Blog IDEA
http://identity-centric-architecture.blogspot.com/2006_12_01_archive.html

CardManager Keys
http://forums.sun.com/thread.jspa?messageID=10025872

Clareity SSO Toolkit
http://code.crt.realtors.org/projects/websso/wiki/WikiStart

Cryptographic Authentification with Java Card
http://www-personal.umich.edu/~lsiden/tutorials/signed-applet/signed-applet.html

Firefox Extensions
http://www.softwaredeveloper.com/features/firefox-extension-resource-072307/

IdentityMeme SAML vs OpenID
http://identitymeme.org/doc/draft-hodges-saml-openid-compare-06.html

JavaScript Form Validation
http://www.javascript-coder.com/html-form/javascript-form-validation.phtml

Jarsigner in java
http://www.onjava.com/pub/a/onjava/2001/04/12/signing_jar.html?page=2

Keystore and Private key
http://www.agentbob.info/agentbob/79-AB.html

Lasso
http://lasso.entrouvert.org/

OpenASelect
http://www.openaselect.org/trac/openaselect/wiki/DevelopingAuthenticationMethods

OpenID4Java
http://code.sxip.com/openid4java/

OpenID Libraries
http://wiki.openid.net/Libraries

OpenID Howitworks
http://www.clickpass.com/docs/howopenidworks
http://www.windley.com/archives/2006/04/how_does_openid.shtml

OpenID Specification
http://openid.net/developers/specs/

OpenSAML
http://www.opensaml.org/

OpenSSO
http://opensso.org/

OReilly Applet communication
http://www.unix.com.ua/orelly/java-ent/servlet/ch10_01.htm

SAML Community
http://saml.xml.org/

SAML IBM myths
http://www.ibm.com/developerworks/xml/library/x-samlmyth.html

SAML Implementations
http://saml.xml.org/wiki/saml-open-source-implementations

SAML 1.0 Toolkit
http://www.sourceid.org/projects/saml_1_1_toolkit.cfm

Signed Applet
http://www-personal.umich.edu/~lsiden/tutorials/signed-applet/signed-applet.html

Shibboleth
http://shibboleth.internet2.edu/

SwitchAAI
http://www.switch.ch/aai/

Verisign JOID
http://code.google.com/p/joid/

Wikipedia
http://en.wikipedia.org/wiki/Federated_identity
http://en.wikipedia.org/wiki/Identity_Metasystem
http://en.wikipedia.org/wiki/Information_Card
http://en.wikipedia.org/wiki/Identity_Selector
http://en.wikipedia.org/wiki/Higgins_trust_framework
http://fr.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol
http://en.wikipedia.org/wiki/List_of_OpenID_providers
http://en.wikipedia.org/wiki/Cardspace
http://en.wikipedia.org/wiki/OpenID
http://fr.wikipedia.org/wiki/OpenID
http://en.wikipedia.org/wiki/SAML
http://en.wikipedia.org/wiki/SAML_2.0
http://fr.wikipedia.org/wiki/Security_assertion_markup_language

X509Certificate conversion
http://www.exampledepot.com/egs/javax.security.cert/ConvertCert.html

X509Certificate to PEM
http://forums.sun.com/thread.jspa?threadID=5130710&messageID=9469397

XML SAML Coverpage
http://xml.coverpages.org/saml.html

12 Annexes

Tous les annexes sont présents sur le CD/DVD et ne sont pas imprimés. La documentation JavaDoc est présente pour chaque version dans les sources.

12.1 Plan du CD/DVD

- Documents
  - Rapport
  - Planning
  - PV des réunions
  - Sources
- Logiciels
- Mesures de sécurité
- Mesures WireShark
- Prototype
  - Version 1.0
    - Binaires
    - Librairies
    - Sources
  - Version 1.1
  - Version 1.2